Query Evaluation: Enumeration, Maintenance, Reliability

Antoine Amarilli

Query Evaluation: Enumeration, Maintenance, Reliability

Habilitation à diriger des recherches

Institut polytechnique de Paris

Antoine Amarilli

Jury:

• **Arnaud Durand**, Professeur, Université Paris-Cité	examinateur
• **Nicole Schweikardt**, Professeure, Humboldt-Universität zu Berlin	examinatrice
• **Thomas Schwentick**, Professeur, Technische Universität Dortmund	rapporteur
• **Luc Segoufin**, Directeur de recherches, INRIA	rapporteur
• **Dan Suciu**, Professeur, University of Washington	rapporteur
• **Sophie Tison**, Professeure émérite, Université de Lille	examinatrice

Soutenue le 4 avril 2023 à Télécom Paris

INSTITUT POLYTECHNIQUE DE PARIS

To my friends and family, to my loved ones, three queries:
May we enumerate many more moments together;
May we maintain the bonds that connect us;
And may we always rely on each other.

Contents

CHAPTER 1

Introduction

This thesis focuses on the *query evaluation* problem of determining whether some data satisfies some requested pattern. I study this problem from the angle of data management, though it also occurs in many areas of computer science: in *string processing*, where we want to find occurrences of a textual pattern; in *streaming algorithms*, where we want to check if a sequence satisfies a specification; in *logic*, where we want to test if a formula holds over a logical structure; in *graph algorithms*, where we look for subgraphs of interest in an input graph; and in many other fields.

Query evaluation can be studied from a practical angle, where the goal is to design efficient implementations and validate them experimentally over benchmarks; but my research concentrates on theoretical results. The goal is then to determine the *computational complexity* of query evaluation: we want to design algorithms in an abstract computation model and establish efficiency guarantees as *upper bounds* on their running time; or we want to show *lower bounds* on how efficient an algorithm can hope to be in such a model, often under some hardness assumptions. The complexity of the problem is a function which depends both on the size of the query and on the size of the data on which we evaluate it. Often, the goal is to achieve the best possible running time as a function of the data, called *data complexity*, while assuming that the query is fixed.

The computational complexity of query evaluation has been abundantly studied, for various kinds of queries over various types of data. Important lines of work in this direction include the study of *monadic second order queries* (or equivalently finite-state automata) over words, over trees, or over relational instances whose *treewidth* is bounded by a constant; the study of *first-order queries* over structurally restricted structures such as *bounded-degree*, *bounded-*

expansion, or *nowhere-dense* structures; and the study of *conjunctive queries* and *unions of conjunctive queries* over arbitrary relational instances.

The evaluation of queries on data can be posed in several different ways. The simplest and most common phrasing is to express it as a *decision problem*: given the query and data, we must determine whether or not the query is satisfied, i.e., a Boolean answer. However, there are more challenging questions that one can ask, given a query and an instance: returning answers; counting answers; highlighting how the query depends on the input data; explaining why the query holds or does not hold; etc. These questions are motivated by practical needs, and they can lead to theoretically relevant problems. The research presented in this thesis focuses on three of these extended forms of query evaluation:

- *Maintenance*: How does the query result change when the underlying data changes? We formalize this as a problem over dynamic data: given a Boolean query and a relational instance, prepare auxiliary data structures that can be used to efficiently determine the status of the query and update it whenever the data is changed. Can we do better than naively reprocessing the whole data after each change?

- *Reliability*: How confident are we that the query is true over uncertain data? We formalize this as a weighted counting problem: given a Boolean query and a relational instance whose facts are annotated by independent probabilities, efficiently determine the probability that the query holds, i.e., the total probability of the possible states of the data that satisfy it. Can we do better than naively summing over all possible worlds?

- *Enumeration*: How can we efficiently compute all results of the query? We formalize this in a specific model of enumeration algorithms: given a query and data, first preprocess the data to prepare auxiliary structures, and then produce a stream of all query answers, without duplicates, while ensuring a small delay between any two consecutive answers. Can we do better than naively testing all possible results (yielding a large delay), or precomputing a full list of the results (yielding a large preprocessing time)?

These three areas have each been studied on their own, and there are also works looking at the intersection of some of them, in particular works studying the incremental maintenance of enumeration structures. This manuscript reviews the state of the art for each of these three problems in the field of data management, presents my research contributions to them, and proposes some directions for further investigation. The hope is to identify connections between these areas, in particular through the unifying theme of *circuits*. Intuitively,

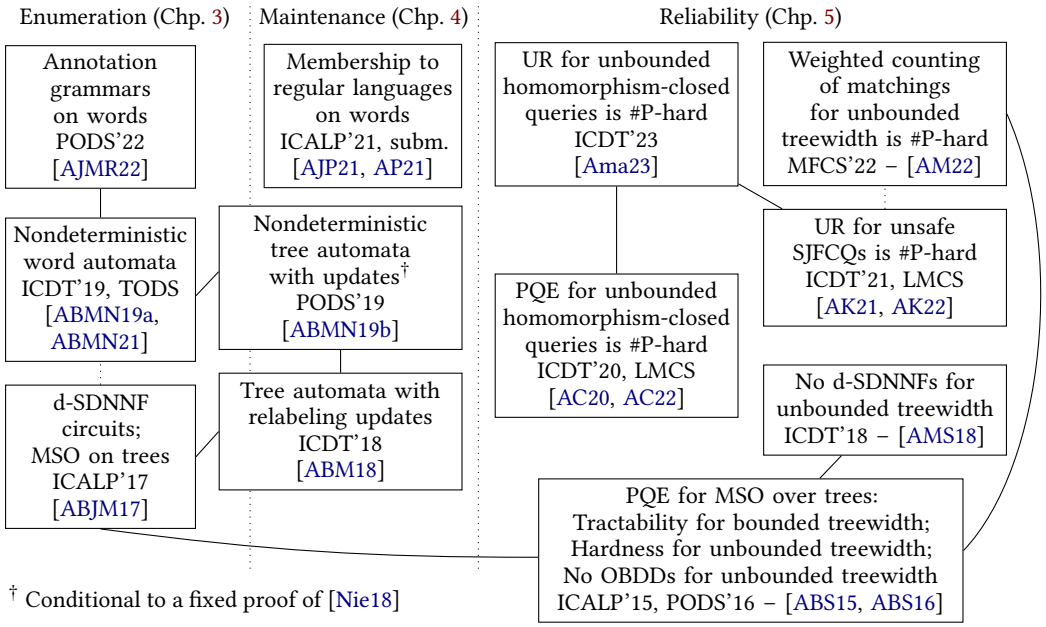

Figure 1: Illustration of the lines of work presented in this manuscript. We omit technical hypotheses such as restrictions to arity-two, use of randomized reductions, constructibility hypotheses, nature of lineage lower bounds, etc.

circuits can be used to express how the query result is computed from the data, in a way which can be useful for all three tasks: they can efficiently recompute the query result when the data is updated; they can count how many subinstances of the data satisfy the query; and they can give a factorized representation of the query results on which we can run an enumeration algorithm.

This manuscript is structured thematically along these three areas of *enumeration* (of query results), *maintenance* (of query results over dynamic data), and *reliability* (of queries over probabilistic data). In the rest of this introduction, I present my research contributions following a chronological perspective: I explain how I have focused on these problems, building on some of my PhD research, and exploring new directions in various collaborations and visits. I also mention the conferences and journals where the results were published. These lines of work are graphically summarized on Figure 1. In the interest of brevity, this manuscript does not cover all the research I have undertaken since my PhD; in particular it omits works that do not fit into one of these three main areas.

PhD work. The first area which I studied is reliability: I investigated during my PhD the task of query evaluation over probabilistic data or *probabilistic query*

evaluation (PQE). Specifically, my PhD studied the evaluation of monadic second-order (MSO) queries over trees and bounded-treewidth relational instances. The input probabilistic data is given in the *tuple-independent database* model where facts are annotated by independent probabilities. The PQE problem for a query asks us compute the probability that the query holds in the resulting distribution. I showed the tractability of this task, via the so-called *intensional approach*: compute a *provenance circuit* (intuitively describing why the data satisfies the query), show that it falls in a tractable class of circuits studied in *knowledge compilation* (specifically, d-SDNNFs), and use this to compute the probability.

As a converse result, I showed that, under some technical assumptions, this probabilistic query evaluation task could not be solved efficiently if the treewidth of the data was unbounded, no matter which other restrictions were imposed on the structure of the input data instances. This was shown in two senses, both relying on the technical tools of extracting grids from high-treewidth instance families [CC16]. First, I showed that the intensional approach could not apply, in the sense that there did not exist small provenance circuit representations in the weaker tractable formalism of *ordered binary decision diagrams* (OBDDs). Second, I showed that the task was intractable by showing a randomized reduction from a #P-hard problem.

These results were a collaboration with my official PhD advisor Pierre Senellart, together with my long-time collaborator and unofficial PhD advisor Pierre Bourhis. They were presented in my PhD thesis, and were published at ICALP'15 [ABS15] for the upper bounds, and at PODS'16 [ABS16] for the lower bounds.

Extending my PhD results on probabilistic query evaluation. After the end of my PhD, one first direction of research was to further extend our intractability results on probabilistic query evaluation. This took place as part of the PhD thesis of Mikaël Monet, whom I co-supervised during his last year together with Pierre Senellart.

We first extended the lower bound on the intensional approach, by showing that some queries over unbounded-treewidth data also did not admit small provenance circuit representations in the more general formalism of d-SDNNFs used in my tractability results. This result was presented at ICDT'18 [AMS18] and in the PhD thesis of Mikaël.

Second, collaborating again with Mikaël some years later, we showed that the hardness result on unbounded-treewidth instances already applied to probabilistic query evaluation for a simpler query, which can be understood as the weighted counting of the matchings of an input graph. This result was presented at MFCS'22 [AM22].

These contributions are presented in Section 5.3, after reviewing the results that were obtained during my PhD.

Using circuits for other tasks. A broader direction, initiated shortly after the end of my PhD, was to extend my circuit-based tractability results on MSO evaluation, and apply them to the area of enumeration, and later of incremental maintenance.

First, in a collaboration with Pierre Bourhis, Louis Jachiet, and Stefan Mengel, we studied how tractable circuit classes from knowledge compilation can be used to enumerate query answers. In particular, combined with the intensional methodology of computing provenance circuits, we could use this result to re-capture the known fact [Bag06, KS13] that query enumeration over trees can be done with linear preprocessing and output-linear delay. These results were published at ICALP'17 [ABJM17]; they are presented in Section 3.3 (describing the enumeration technique via circuits) and Section 3.4 (explaining how to recapture the result for MSO evaluation).

Second, we realized that this circuit-based enumeration technique, together with the computation of balanced tree decompositions [BH98], could be used on dynamic data to efficiently maintain enumeration structures. Specifically, we show that, for any fixed MSO query, given an input tree, we can preprocess it in linear time to construct an output-linear delay enumeration structure, and maintain the structure under relabeling updates to the tree in logarithmic time, improving on previous results [LM14]. This was published at ICDT'18 [ABM18] and is presented in Section 4.3.

Third, we noticed how these results could be combined with a result of Niewerth [Nie18] on the incremental maintenance of balanced tree representations. The goal was to use this result to keep the tree balanced under more expressive updates, namely, leaf insertion and deletion. This was presented at PODS'19 [ABMN19b]. Unfortunately, it later appeared that the proof of the incremental balancing result [Nie18] was flawed; the result of [ABMN19b] on incremental maintenance only holds if the result of [Nie18] can be correctly proved. A corrected proof of [Nie18] was recently announced in a preprint [KMMN22].

Enumeration for document spanners. The tools developed with Pierre, Stefan, and Matthias for MSO enumeration turned out to be useful in the setting of *document spanners*, a formalism for declarative information extraction in database theory [FKRV15]. Regular spanners are subsumed by MSO in terms of expressiveness, but an interesting question is whether their matches can be tractably enumerated in *combined complexity*, i.e., taking also into account the size of the spanner. It had been shown earlier [FRU+18] that the matches of

document spanners on words could be enumerated with linear preprocessing and output-linear delay while remaining tractable in combined complexity; however, this assumed that the input spanner was represented as a *deterministic* variable-set automaton, implying an exponential blowup if the input spanner is not already deterministic.

We accordingly showed that the same bounds can be achieved with a polynomial dependency on the input variable-set automaton, even if it is nondeterministic. In particular, this meant that the same result applies to other ways to express regular spanners, e.g., regex-formulas. This work was published at ICDT'19 [ABMN19a] and was featured in ACM SIGMOD Research Highlights; it is presented in Section 3.5.

Later, I welcomed Cristian Riveros's PhD student Martín Muñoz on a visit to work with Louis Jachiet and myself, and we studied how these results on enumeration algorithms could be extended beyond regular spanners. Specifically, we considered a formalism inspired by context-free grammars, dubbed *annotation grammars*. A slightly different extension of document spanners to context-free languages had been investigated shortly beforehand by Peterfreund [Pet21], with an enumeration algorithm ensuring polynomial-time preprocessing and constant delay in data complexity. We were able to improve on these bounds, and show enumeration with cubic preprocessing in the document and output-linear delay. Further, we showed that the preprocessing time could be made quadratic or even linear for some restricted classes of annotation grammars. This work was published at PODS'22 [AJMR22] and is presented in Section 3.6.

Incremental maintenance. After studying the incremental maintenance of enumeration structures of MSO queries on trees, I turned to the simpler question of incremental maintenance for Boolean MSO queries (without enumeration), on words and under relabelings. In other words, fixing a regular language, the question is to maintain the information of whether a string belongs to the language, under substitution updates. This was a collaboration with Charles Paperman (who brought expertise in algebraic automata theory) and Louis Jachiet (who brought expertise on machine models and data structure lower bounds). We showed a conditional trichotomy over regular languages for the complexity of this problem, improving on earlier work [SFMS97]. This was presented at ICALP'21 [AJP21], where it received the best paper award of ICALP'21 track B; it depends on an auxiliary article with Charles which is currently under review [AP21]. This work is presented in Section 4.4.

Back to probabilistic databases. Some years after my PhD, I came back to the question of probabilistic query evaluation and investigated it under new

angles. First, visiting İsmail İlkan Ceylan in Oxford, we started collaborating on the evaluation of recursive queries over probabilistic data. Specifically, we focused on the class of *homomorphism-closed* queries that are *unbounded* in the sense that they are not equivalent to a union of conjunctive queries. This query class covers in particular regular path queries and Datalog queries. We were able to show that probabilistic query evaluation (PQE) is intractable for *all* such queries, restricting for technical reasons to arity-two signatures. This work was published at ICDT'20, where it received the best paper award; it is presented in Section 5.4.

Second, visiting Benny Kimelfeld in Technion, we studied the problem of *uniform reliability* (UR) for queries. This is a restriction of the PQE problem where we impose that all facts of the input probabilistic databases have probability 1/2; equivalently, the task is to count how many subinstances of the input database satisfy the query. We showed that, for conjunctive queries without self-joins, the tractability boundary is the same as for PQE, i.e., the problem is tractable if and only if the query is hierarchical [DS07]. We published this result at ICDT'21 [AK21]. Inspired by this result, I then studied uniform reliability for unbounded homomorphism-closed queries: I showed that the intractability of PQE on arity-two signatures can be lifted to UR. This work will be published at ICDT'23 [Ama23]. My contributions to the study of the UR problem are presented in Section 5.5.

Structure of the manuscript. This manuscript is structured along the three axes of enumeration, maintenance, and reliability, presented in three chapters. Before this, I first give formal definitions of some required preliminary notions (Chapter 2): the goal is to ensure that the manuscript can be read in a somewhat self-contained way, so these preliminaries can be skipped by readers who are sufficiently familiar with the concepts or only interested in the high-level picture.

I then present the areas of enumeration algorithms (Chapter 3), maintenance over dynamic data (Chapter 4), and probabilistic query evaluation (Chapter 5). Each of these three chapters follows the same structure. First, I start with an introduction that surveys the main lines of research in this area, focusing on data management and neighboring communities. I then give a short overview of my research contributions in the area. Then, I present the contributions in more detail in separate sections: I give their context, their prerequisites, and their formal statement, before sketching some of the technical details. Each chapter is concluded by giving my perspectives for further research in the area.

CHAPTER 2

Preliminaries

We give here formal definitions for some technical notions used in several places of this manuscript.

Relational instances. A *relational signature* σ is a set of *relation names* each associated to an *arity* in $\mathbb{N}_{>0}$. We call σ *arity-two* if the arity of every relation is 1 or 2. A *relational instance* I over σ is a set of facts of the form $R(\vec{a})$ where R is a relation name of σ and $\vec{a}$ is a tuple of constants whose arity is that of R. Facts over one elements are called *unary*, and facts over two elements are called *binary*. The *domain* $\mathrm{dom}(I)$ of I is the set of constants that occur in I. A *subinstance* I' of I, written $I' \subseteq I$, is simply a subset of the facts of I; we then have $\mathrm{dom}(I') \subseteq \mathrm{dom}(I)$.

The *Gaifman graph* of I is the undirected graph whose vertex set is the domain of I, and where there is an edge between two constants whenever they co-occur in a fact of I. We will use some standard measures on instances, such as their *treewidth*, which is defined as that of their Gaifman graph. This is defined via *tree decompositions*: a *tree decomposition* of an undirected graph $G = (V, E)$ is a tree T with a function μ mapping each node n of T to a subset $\mu(n)$ of V such that two conditions are satisfied: (i.) for any vertex $v \in V$ of G, its occurrences $\{n \in T \mid v \in \mu(n)\}$ in T form a connected subtree of T; and (ii.) for every edge $\{u, v\} \in E$, there is a node n of T such that $\{u, v\} \subseteq \mu(n)$. The *width* of T is the maximal cardinality of an image of μ minus 1, i.e., $\max_{n \in T} |\mu(n)| - 1$. The *treewidth* of G is then the minimal width of a tree decomposition of G. For details, see, e.g., [Bod94].

A *homomorphism* from a σ-instance I to another σ-instance I' is a function h from the domain of I to the domain of I' such that, for each fact $F = R(\vec{a})$ of I,

the fact $R(h(\vec{a}))$ is a fact of I', where we denote by $h(\vec{a})$ the vector obtained by mapping each element of $\vec{a}$ according to h.

Queries. A *(Boolean) query* Q over a relational signature σ is a Boolean function on instances over σ, i.e., an instance can either *satisfy* or *violate* the query. We will use several query languages in this manuscript, in particular *conjunctive queries* (CQs), *unions of conjunctive queries* (UCQs), *first-order logic* (FO), and *monadic second-order logic* (MSO). We introduce them in more detail below.

A *conjunctive query* (CQ) over the relational signature σ is an existentially quantified conjunction of atoms over σ. Formally, it is an expression of the form $Q(\vec{x}) : \exists \vec{y}\ A_1 \wedge \cdots \wedge A_n$ where the $A_1, \ldots, A_n$ are *atoms* over σ, i.e., expressions of the form $R(\vec{z})$ with R a relation name of σ and $\vec{z}$ a tuple of variables from $\vec{x}$ and $\vec{y}$ whose arity is that of R in σ. The variables $\vec{x}$ are the *free variables*: we call the query *Boolean* if it has no free variables.

A Boolean CQ Q *holds* on an instance I if there is a *homomorphism* from Q to I, i.e., a function from the variables used in the atoms to the constants occurring in I such that the image of each atom is a fact of I. If $Q(\vec{x})$ is non-Boolean, given a tuple $\vec{a}$ of constants of $\mathrm{dom}(I)$ whose arity is that of $\vec{x}$, we call $\vec{a}$ an *answer* (or *result*) of Q on I if I satisfies the Boolean CQ with constants $Q(\vec{a})$ obtained from Q by substituting the free variables by $\vec{a}$. In other words, $\vec{a}$ is an answer if there is a homomorphism from Q to I that maps $\vec{x}$ to $\vec{a}$.

A *self-join-free CQ* (SJFCQ) is a CQ that does not use the same relation name from σ in two different atoms. A *connected CQ* is a CQ Q having a connected Gaifman graph, where the Gaifman graph of Q is that of the relational instance defined from Q by seeing variables as constants in the expected way. A *CQ with inequalities* ($\mathrm{CQ}^{\neq}$) is a CQ where we additionally allow *inequality atoms* of the form $x \neq x'$ where x and x' are variables, with the requirement that every variable occurs in an least one atom which is not an inequality atom.

A *union of conjunctive queries* (UCQ) is a finite disjunction of CQs that all have the same free variables. Its set of *answers* on an instance I is the union of the answers of each of its constituent CQs on I. The UCQ is *Boolean* if all its constituent CQs are, in which case it is *satisfied* on an input instance if and only one of its constituent CQs is. A *connected UCQ* is one where all constituent CQs are connected. A *UCQ with inequalities* is defined using $\mathrm{CQ}^{\neq}$ queries instead of CQs.

Fixing a signature σ, a *query in first-order logic (FO)*, often also called an *FO formula*, is an expression built from atoms over σ (in which we allow variables and constants), using the operators of conjunction, disjunction, and existential and universal quantification. We can in particular see CQs and UCQs as restricted

classes of FO queries. We omit the definition of what it means for an instance I to satisfy an FO formula; see for instance [Lib04].

Monadic second-order logic (MSO) is the logic that extends FO with quantification over sets. MSO can express some queries that are *recursive*, e.g., "there are two elements x and y such that the facts $R(x)$ and $T(x)$ hold and there is a path of S-facts from x to y".

Queries in FO and MSO can be *Boolean*, or can have free variables. In the case of MSO queries, the free variables may all be first-order, or some may be second-order. In the latter case we assume without loss of generality that all free variables are second-order, as these are more expressive. Given a formula $\phi(\vec{x})$ with free variables and a tuple $\vec{a}$ of constants whose arity is that of $\vec{x}$, we denote by $\phi(\vec{a})$ the Boolean query obtained by replacing the free variables by the constants provided. We define $\phi(\vec{A})$ in the same way when the free variables of $\phi(\vec{X})$ are second-order and $\vec{A}$ is a tuple of sets of constants.

Word, trees, automata. In addition to queries on relational structures, we will also consider the special case of query evaluation on *words* and *trees*. We also define the notion of *finite automata* on these structures as an alternative way to express Boolean MSO queries. All automata considered are finite.

Fixing a finite *alphabet* Σ, a *word* is a finite sequence of characters of Σ. We write Σ^* for the set of all words over Σ. We write $|w|$ for the *length* of $w \in \Sigma^*$, and write ϵ the *empty word*, with $|\epsilon| = 0$.

A *(finite) automaton on words* $A = (Q, \Sigma, I, F, \delta)$ consists of a finite set Q of *states*, subsets $I \subseteq Q$ and $F \subseteq Q$ of *initial states* and *final states*, and a *transition relation* $\delta \subseteq Q \times \Sigma \times Q$. An *run* of A over a word $w = a_1 \dots a_n$ of Σ^* is a sequence $q_0, \dots, q_n$ that starts at an initial state, i.e., $q_0 \in I$, and follows transitions of the automaton, i.e., $(q_{i-1}, w_i, q_i) \in \delta$ for all $1 \leq i \leq n$. The run is *accepting* if it ends at a final state, i.e., $q_n \in F$: we then say that w is *accepted* by A. The *language* of A is the subset of the words of Σ^* that are accepted by A. The automaton A is *deterministic* if it has one initial state, i.e., $|I| = 1$, and if the relation δ is a partial function from $Q \times \Sigma$ to Q.

Having fixed the alphabet Σ, a Σ*-tree* is a binary tree which is *full* (each node has either 0 or 2 children) and where each node is labeled by an element of Σ. A (bottom-up) *tree automaton* $A = (Q, \Sigma, \iota, F, \delta)$ consists of a finite set Q of *states*, an *initial relation* $\iota \subseteq \Sigma \times Q$ giving possible states for leaf nodes depending on their label, a *transition relation* $\delta \subseteq Q \times Q \times \Sigma \times Q$ giving possible states for an internal node depending on the state of its children and on its label, and a subset $F \subseteq Q$ of *final states*. A *run* of A over a Σ-tree T is a function ρ mapping each node of T to a state of Q such that, for every leaf n of T labeled by a, we have $(a, \rho(n)) \in \iota$, and for every internal node n of T labeled by a with child

nodes n_1 and n_2, we have $(\rho(n_1), \rho(n_2), a, \rho(n)) \in \delta$. The run is *accepting* if the root n_0 of T is mapped to a final state, i.e., $\rho(n_0) \in F$: we then say that T is *accepted* by A, and the language of A is again the set of Σ-trees that it accepts. The automaton A is *deterministic* if ι and δ are partial functions respectively from Σ to Q and from $Q \times Q \times \Sigma$ to Q.

We can see automata over words and trees as *Boolean queries* over words on trees, which are satisfied precisely on the words and trees that they accept. Alternatively, we will sometimes talk of FO or MSO formulas over words or trees on an alphabet Σ. These can be defined as formulas over a relational signature σ_Σ consisting of $|\Sigma|$ unary relations indicating the label in Σ of each word position or tree node, and of a binary relation denoting the successor of each position (for words), or the left and right child of each internal node (for trees). We then only consider query evaluation for such formulas on relational instances over σ_Σ that represent words or trees, i.e., where the relations are interpreted in a way that actually describes a word or tree. It is then known that Boolean MSO queries on words have the same expressive power as word automata (by the Büchi-Elgot-Trakhtenbrot theorem), and that Boolean MSO queries on trees have the same expressive power as tree automata [TW68].

Query evaluation. The *(Boolean) query evaluation problem* for a query Q over a relational signature σ is the problem of deciding if a query holds on an instance over σ. The name of *query evaluation* is chosen to ensure a unified terminology, even though it is only standard in database theory and other names are used in other communities: e.g., the problem is typically called *model checking* when the query is expressed as a logical formula, or *evaluating an automaton* when Q is expressed as an automaton.

Formally, query evaluation for Boolean queries is formalized by specifying a class $\mathcal{Q}$ of allowed queries and a class $\mathcal{I}$ of allowed instances. Typical choices for $\mathcal{Q}$ include: all MSO queries over Σ-trees for some alphabet Σ, all CQs over a relational signature σ, all CQs satisfying some property, etc. For the instances $\mathcal{I}$, we can allow, e.g., all relational instances over signature σ, or the relational instances having treewidth $\leq k$ for some constant $k \in \mathbb{N}$, or all Σ-trees, or all words over Σ, etc. An input to the query evaluation problem for $\mathcal{Q}$ and $\mathcal{I}$ then consists of a query Q in $\mathcal{Q}$ and of an instance I in $\mathcal{I}$. The output is the Boolean information of whether I satisfies Q.

The query evaluation problem can also be studied for non-Boolean queries, in which case the task is to determine all answers of the query, i.e., all tuples $\vec{a}$ of values of $\mathrm{dom}(I)$ such that $Q(\vec{a})$ holds on I. For MSO queries with free second-order variables, we must compute all tuples $\vec{A}$ of subsets of $\mathrm{dom}(I)$ such that the query $Q(\vec{A})$ holds.

We can study the complexity of query evaluation in *combined complexity*, where both the query and instance are given as input; or in *data complexity*, where the query (and thus the signature) are fixed, and the complexity is only measured as a function of the instance. The data complexity perspective is somewhat reminiscent of the study of *parameterized complexity* [FG10]; it is common in database theory, where it is motivated by the fact that database instances are typically much larger than queries in practice.

Boolean circuits. Given a set X of *variables*, a *valuation* of X is a function $\nu \colon X \to \{0, 1\}$ mapping each variable of X to a Boolean value. A *Boolean function* over X is a function mapping valuations over X to a Boolean value. We will consider three ways to represent Boolean functions: *Boolean formulas*, *Boolean circuits*, and *binary decision diagrams*.

A *Boolean formula* over a set X of *variables* is inductively defined from the variables of X using the Boolean operations of conjunction, disjunction, and negation. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of disjunction of *literals*, i.e., variables or negations of variables; it is in *disjunctive normal form* (DNF) if it is a disjunction of conjunction of literals. It is *monotone* if it does not feature negation.

A *Boolean circuit* C over X consists of a directed acyclic graph (G, W) whose vertices G are *gates* and whose edges $W \subseteq G \times G$ are *wires*, along with a distinguished *output gate* $g_0 \in G$ and a labeling of the gates which we now present. The gates of G can be *variable gates* labeled with a variable of X, or *internal gates* labeled with one of the Boolean operations of conjunction, disjunction, and negation. The *inputs* of a gate $g \in G$ are the gates g' having an edge to g, i.e., $(g', g) \in W$; and the *fan-in* of g is its in-degree, i.e., its number of inputs. We require that variable gates have fan-in zero and that negation gates have fan-in 1. A circuit is *monotone* if it has no negation gates.

The *evaluation* of a Boolean formula under a valuation ν of its variables is defined in the expected way, by substituting the variables by their value according to ν and evaluating the Boolean operations. The *evaluation* of a Boolean circuit is defined in the same way, by considering the value of the output gate; note that conjunction gates and disjunction gates with a fan-in of zero always evaluate to the neutral element of their operation, i.e., 1 and 0 respectively.

We now introduce some standard restricted circuit classes from the field of knowledge compilation [DM02]. A *negation normal form circuit* (NNF) is a Boolean circuit where negation is only applied to variable gates, i.e., it can be seen as a monotone circuit over literals. A *decomposable NNF circuit* (DNNF) is a Boolean circuit in NNF where there is no variable gate g having a directed path

to two distinct input gates g_1 and g_2 of some conjunction gate g'. In other words, for every conjunction gate g', the subsets of variables $X_1, \ldots, X_n$ of X having a directed path to the respective inputs $g_1, \ldots, g_n$ of g' are pairwise disjoint.

A *v-tree* over the variables X is a rooted unranked ordered tree T whose leaves are in bijection with X. A *structuring mapping* into T for a circuit C is a function λ that maps the variable and conjunction gates of C to T and obeys three conditions: (i.) each variable gate g is mapped to the leaf $\lambda(g)$ of T associated to the variable that labels g; (ii.) each conjunction gate is mapped to an internal node of T; (iii.) the mapping is compatible with the circuit, i.e., if a gate g has a non-empty directed path to another gate g' then $\lambda(g)$ is a strict descendant of $\lambda(g')$ in T. A *structured DNNF circuit* (SDNNF) is an NNF circuit C having a structuring mapping into some v-tree T. Note that a SDNNF is a DNNF, i.e., it is always decomposable; but some DNNFs are not SDNNFs, e.g., the circuit corresponding to $(x \wedge (y \wedge z)) \vee ((\neg x \wedge \neg y) \wedge (\neg z))$.

A *deterministic DNNF circuit* (d-DNNF) is a DNNF circuit C where, for each valuation ν of the variables, for every disjunction gate g, there is at most one input gate of g that evaluates to 1 under ν. We will also consider *deterministic SDNNF circuits*, which we call d-SDNNFs.

A *binary decision diagram* over a set of variables X is a directed acyclic graph D with two distinguished *sink nodes* 0 and 1, and a distinguished *starting node*. The two sink nodes have no outgoing edges, and each other node is labeled with a variable of X and has precisely two outgoing edges respectively labeled 0 and 1. Given a valuation ν of X, the evaluation of D under ν is given by the path which begins at the starting node, ends at a sink node that gives the result of the evaluation, and follows for each traversed non-sink node n one of the two outgoing edges according to the value $\nu(x)$ of the variable x that labels n.

A *free binary decision diagram* (FBDD) is one where no path from the starting node to a sink traverses two nodes labeled with the same variable. An *ordered binary decision diagram* (OBDD) is an FBDD where there is a total order $X = x_1, \ldots, x_n$ on the set X of variables satisfying the following: for every path from the starting node to a sink, the sequence of the node labels is a subsequence of $x_1, \ldots, x_n$.

Provenance circuits. We will use the important tool of *provenance* of queries over instances [GKT07]. The *provenance* of a Boolean query Q on an instance I is the Boolean function ϕ defined on the set of variables I (i.e., its variables are the facts of I), with the following semantics: for every subinstance $I' \subseteq I$, letting ν be the valuation of I that maps each fact $F \in I$ to 0 if $F \notin I'$ and to 1 if $F \in I'$, then ϕ evaluates to 1 under ν if and only if I' satisfies Q. In other words, ϕ evaluates to 1 precisely on the valuations that correspond to a subinstance of I

that satisfies Q. In particular, I satisfies Q if and only if ϕ evaluates to true under the valuation mapping each fact of I to 1.

A *provenance circuit* for Q on I is then simply a Boolean circuit representing the provenance of Q on I. Note that the computation of provenance circuits is a task which generalizes query evaluation, as we can use them in particular to determine whether the query is satisfied. Further, provenance circuits can sometimes be computed by adapting Boolean query evaluation algorithms. As an example, given a CQ Q and relational instance I, we can build a provenance circuit for Q on I, more precisely as a Boolean formula in DNF, simply by writing one disjunct for each mapping from the atoms of Q to the facts of I. This takes time $O(|I|^{|Q|})$, i.e., it is polynomial in data complexity.

The study of provenance circuits is motivated by the practical question of *data provenance* in databases [GKT07], which was originally phrased using Boolean formulas before being extended to circuit representations [DMRT14]. It was also recently studied for logical formulas outside of the context of databases [DGNT21], and it can be further generalized beyond the Boolean semiring [GKT07], though we will not explore this in this manuscript. One important question about provenance circuits is whether we can efficiently compute circuits falling in the restricted classes introduced above, e.g., d-DNNFs, OBDDs, etc.

Computational model. The complexity results presented in this manuscript assume the abstract model of computation formalized as the *random access machine* (RAM), with unit cost model. Intuitively, the memory of the machine consists of cells that can hold integers whose size is logarithmic in the input data; we can use them in arithmetic operations, or use them as pointers to access memory cells. We assume that the arithmetic operations take constant time, even though the values to which they apply are of logarithmic size; this is motivated by the practical design of computers where arithmetic operations on registers are typically assumed to take constant time. Further, we assume that pointers can be dereferenced in constant time: this is again motivated by the design of practical computers, and differs, e.g., from Turing machines.

For a recent formalization of the RAM model and of its expressive power, see the work by Grandjean and Jachiet [GJ22]. Some of the results reviewed in this manuscript further assume that the machine can use 2-dimensional arrays, which may strictly extend the expressive power of the model (see [GJ22, Open problem 1]).

CHAPTER 3

Enumeration Algorithms for Query Evaluation

This chapter focuses on the query evaluation task of *enumeration*, where we want to efficiently list the results of a non-Boolean query in a streaming fashion. I present the context of enumeration algorithms, focusing on the setting of data management, before presenting my contributions and some research perspectives.

3.1 Introduction

Enumeration algorithms. The study of computational complexity usually focuses on *decision problems*, where the answer to compute is Boolean, and the complexity is measured as a function of the input. However, this formalism breaks down when the output is non-Boolean and can be much larger than the input data. For example, consider the task, given an OBDD, of listing all its satisfying assignments. If the OBDD always evaluates to 1, there are exponentially many assignments to return. Thus, the worst-case complexity of an algorithm for the problem must be exponential as a function of the input. This means that we cannot beat the naive algorithm that tests the truth value of every possible assignment, at least not according to the metric of worst-case complexity as a function of the input.

Hence, in such settings, we need to measure the complexity in a different way. There are several options, e.g., considering average-case complexity, summarizing the output (e.g., count the number of satisfying assignments), representing the output in factorized form, etc. Our focus in this chapter is

on *enumeration algorithms*, where we produce the full output and measure the worst-case complexity in a different way.

Enumeration algorithms are a refinement of the well-studied notion of *output-sensitive algorithms*, where we measure the complexity as a function of the input *and* of the output. For instance, the naive algorithm to list satisfying assignments is not an efficient output-sensitive algorithm, as it will always take exponential time, even if the formula is unsatisfiable and the output is empty. However, the guarantee on output-sensitive algorithms only applies to the running time needed to produce the *entire output*: if there are many results, it may take very long before the first result is produced, or the *delay* between results can sometimes be very large. *Enumeration algorithms* thus refine output-sensitive algorithms to enforce stronger guarantees on the running time when we want to produce multiple results as a sequence.

Formally, when considering a problem whose output is a set of results, an *enumeration algorithm* reads the input and produces the results in succession, one after the other, and without repetition. Its performance is measured along two axes: the *preprocessing time*, which is the time taken to produce the first result (or conclude that there are none), expressed as a function of the input; and the *delay*, which is the maximal time taken to produce any given result after having produced the previous one, expressed as a function of the input and of the size of each result. We focus here on *worst-case delay*, though the notion of *average delay* (in an amortized sense) has also been investigated. In terms of memory usage, we can measure the memory size used after preprocessing, i.e., when producing the first result; and the growth of the memory usage as the results are enumerated. However, we will generally not consider the question of memory usage for the results presented in this manuscript.

Enumeration algorithms for data management. The study of output-sensitive and enumeration algorithms in computer science is not new: they have been investigated for decades, e.g., to enumerate combinatorial structures [Val79] (see [Rus03]), or to list structures in graphs such as cliques [TIAS77] or spanning trees [KR95]. This research forms a natural counterpart to the study of streaming algorithms [Mut05] which studies how to *consume* data streams. For a general survey of problems for which enumeration algorithms have been studied, see for instance [Was16]. Common measures of tractability in these works are *output-polynomial running time*, or *polynomial preprocessing and delay*. In the latter, the first solution, and each successive solution, can be produced in polynomial time in the input (and typically has polynomial size in the input).

In the field of data management, the study of enumeration algorithms can sometimes aim for better bounds than polynomial preprocessing and delay.

One very stringent requirement is that of *constant delay*, or *output-linear delay* [Bag06]. In this case, we require that the enumeration phase produces each result in time linear in its size, i.e., in constant delay when the results have constant size; which is unavoidable if the outputs must be written in full. Note that this implies that the delay does *not* depend on the size of the input to the enumeration algorithm. Also note that our notion of output-linear delay in this manuscript is somewhat less restrictive than that of [Bag06] because we do not bound the memory usage, unlike [Bag06]. As for the preprocessing, the most stringent requirement is *linear preprocessing* (as a function of the input): this is often unbeatable, because it is often necessary in the worst case to read the entire input to find out if there are results to produce. We phrase these requirements in *data complexity*, i.e., the query is assumed to be fixed.

Enumeration algorithms with such strong requirements have been investigated in database theory for the task of query evaluation in various settings. In particular, early works on the topic have studied the enumeration of MSO query results over trees: see Bagan [Bag06] and Kazana and Segoufin [KS13], who established that this could be done with linear preprocessing and output-linear delay; and Courcelle [Cou09]. Another early line of work has focused on conjunctive queries (CQs) on arbitrary relational instances: Bagan, Durand, and Grandjean [BDG07] showed that enumeration could be achieved with linear preprocessing and constant delay in data complexity for the class of *acyclic free-connex* CQs, and showed (conditionally) that this was not possible for self-join-free CQs that were not in this class. Enumeration was also studied for first-order (FO) formulas on bounded-degree structures: it was shown by Durand and Grandjean [DG07], and re-proved by Kazana and Segoufin [KS11], that the task could be solved with linear preprocessing and constant-delay data complexity. These early results were surveyed by Segoufin in an invited talk at EDBT/ICDT'13 [Seg13], at STACS'14 [Seg14], and in SIGMOD Record [Seg15].

Since then, enumeration has been a very active area of research in data management. For instance, for FO queries, Segoufin and Vigny generalized to enumeration the results on FO model checking on databases with *local bounded expansion* [SV17] and later with Schweikardt on *nowhere dense graphs* [SSV22]. For CQs, the tractability of enumeration for acyclic free-connex CQs [BDG07] was extended by Berkholz and Schweikardt to CQs of bounded free-connex submodular width [BS19] with *FPT-preprocessing time* (which in particular ensures polynomial data complexity) following the corresponding results on CQ evaluation shown by Marx [Mar13], and generalizing similar results on *free-connex treewidth* already present in Bagan, Durand, and Grandjean [BDG07]. The results about CQs were extended to UCQs first by Berkholz, Keppeler, and Schweikardt [BKS18b], and then by Carmeli and Kröll [CK21], leaving the picture still incomplete. Carmeli and Kröll also investigated enumeration

algorithms on database schemas featuring functional dependencies [CK20].

The activity around enumeration topics also touches on neighboring areas, e.g., the study of enumeration complexity classes [CS19], the enumeration of models of logical formulas [CS21], or enumeration for ontology-mediated queries [LP22]. Two other important research directions are that of enumeration algorithms for *document spanners*, which I present in more detail in Section 3.5 below; and incremental maintenance of enumeration structures on *dynamic data*, which I present in the next chapter. Recent surveys on the topic of enumeration include the one by Strozecki [Str19] or, for the task of evaluating CQs, the tutorial by Berkholz et al. [BGS20].

3.2 Structure of the Chapter

This chapter presents my research contributions on the task of enumerating query answers, which all rely (explicitly or implicitly) on circuit-based methods.

The first section (Section 3.3) presents our work with Pierre Bourhis, Louis Jachiet, and Stefan Mengel, about enumeration tasks on circuit classes from knowledge compilation [ABJM17]. We study the problem of enumerating the satisfying assignments of circuits, in a model of *set circuits in d-DNNF* related to knowledge compilation and factorized representations. We show how the satisfying assignments of such set circuits can be enumerated with linear preprocessing and delay which is output-linear, i.e., linear in the size of each produced assignment. We further show that how Boolean circuits can be translated to set circuits, so our algorithms can be used to enumerate the satisfying assignments of Boolean circuits in the restricted d-SDNNF class from knowledge compilation.

The second section (Section 3.4) presents our work on enumerating the results of MSO queries on trees. We show how we can efficiently construct d-SDNNF circuits which capture the results of such queries. Thus, applying the previous result on circuits [ABJM17], we can recapture the result by Bagan [Bag06] and Kazana and Segoufin [KS13] that the results of MSO queries on trees can be enumerated with linear preprocessing and output-linear delay. These results were mentioned in [ABJM17] as an application of enumeration via circuits.

The third section (Section 3.5) restricts from the setting of trees to that of words, and explains how we can efficiently enumerate the results of *document spanners*, a formalism for declarative information extraction. In terms of data complexity, these results follow from those of the previous section, because regular spanners can be expressed in MSO over words. However, we show that we can also ensure tractable *combined complexity* [ABMN19a], i.e., additionally enforce tractability in the representation of the spanner.

In the fourth section (Section 3.6), we present our recent extension [AJMR22] of regular spanners called *annotation grammars*, where we replace finite automata with context-free grammars as a formalism to specify the extraction task. We show how to efficiently enumerate the mappings of such a grammar on an input word with cubic preprocessing and output-linear delay, improving on a quintic algorithm for a similar formalism in [Pet21]. We also show quadratic and even linear preprocessing for some restricted classes of grammars.

3.3 Efficient Enumeration via Knowledge Compilation

My first contribution to the field of enumeration algorithms was to study how to enumerate the *satisfying assignments* of Boolean circuits [ABJM17]. This is a collaboration with Pierre Bourhis and Stefan Mengel, started shortly after my PhD.

The problem of enumerating satisfying assignments had been studied previously in the context of Boolean formulas [JLSS14] and circuits [JS05], but the motivation there was mainly practical. Indeed, from a theoretical angle, given an arbitrary Boolean formula or circuit, it is NP-hard even to determine if some satisfying assignment exists. Our approach in [ABJM17] was to achieve efficient upper bounds by focusing on circuits in tractable classes from knowledge compilation; in particular d-SDNNF circuits, for which the satisfiability problem is easy to solve.

The enumeration of satisfying assignments for such circuits is a natural question, but our motivation was also to introduce a *modular approach* for the design of enumeration algorithms, also inspired by factorized representations in databases [OZ15]. Indeed, as we reviewed in the introduction of this chapter, there are many problems for which we can study enumeration. Instead of devising enumeration algorithms for each problem from scratch, we believe that it would be more convenient to proceed in three steps. First, explain how to efficiently compute a circuit that gives a factorized representation of the solutions to enumerate. Second, show that this representation falls in one of the tractable classes from knowledge compilation [DM02]. Third, simply invoke a general-purpose enumeration algorithm on such representations (e.g., the one we propose). We believe that this approach is promising because it neatly decouples the techniques used for query evaluation (first step) from the complicated bookkeeping required for efficient enumeration (third step), using well-known circuit classes as an interface between the two (second step).

We specifically focus on the class of *d-SDNNF* circuits, and our work shows

that we can efficiently enumerate their satisfying assignments:

Theorem 3.3.1 ([ABJM17], Theorem 2.1)**.** *Given a d-SDNNF circuit and a v-tree that structures it, we can enumerate its satisfying assignments with linear preprocessing and output-linear delay.*

Our result also gives a bound on the memory usage (which unfortunately depends on the size of the input, unlike delay), and an efficient algorithm if we want to enumerate only the assignments of a given Hamming weight. We sketch the proof of Theorem 3.3.1 in the remainder of this section.

Proof step 1: Converting to set circuits. The first step of our proof, and in fact the only point where we use the structuredness assumption, is to convert the input circuit to a different formalism of *set circuits*, that features only implicit negation. These are defined like Boolean circuits, but with other internal gates and with a different semantics:

Definition 3.3.2 (adapted from [ABMN19b])**.** *Given a set X of variables, an* assignment *is simply a subset of X. A* set circuit *over X is a circuit C built of* variable gates *(each labeled by a variable of X) and operators $\times$ (having exactly two input gates) and $\cup$ (having at least one input gate). Each gate g* captures *a set $\mathrm{S}(g)$ of assignments:*

- *a variable gate g labeled with variable x captures the singleton set $\mathrm{S}(g)$ containing only the assignment $\{x\}$;*
- *a product gate g captures the relational product of its inputs g_1 and g_2, i.e., $\mathrm{S}(g) = \{s_1 \cup s_2 \mid s_1 \in \mathrm{S}(g_1), s_2 \in \mathrm{S}(g_2)\}$;*
- *a union gate g captures the union of its inputs $g_1, \ldots, g_n$, i.e., $\mathrm{S}(g) = \bigcup_{1 \leq i \leq n} \mathrm{S}(g_i)$.*

Note that the set captured by a gate can never be empty, and that captured sets also cannot contain the empty assignment. The set of assignments captured *by C is $\mathrm{S}(g_0)$ where g_0 is the distinguished output gate. We say that C is* in d-DNNF *if:*

- *the input of $\times$-gates are always on disjoint domains, formally, there is no variable gate g having a path in the circuit to two gates g_1 and g_2 that are the two inputs of a $\times$-gate;*
- *the unions in $\cup$-gates are always disjoint.*

Notice that set circuits do not feature negation, but their semantics is subtly different from that of monotone Boolean circuits. In Boolean circuits, the absence of a variable means that it can be indifferently true or false; in set circuits, the missing variables are implicitly required to be false. This mechanism of implicit negation has also been studied for OBDDs where it is called the *zero-suppressed semantics* [Weg00, Chapter 8].

Example 3.3.3. *In a Boolean circuit C corresponding to the Boolean formula $x \vee (y \wedge z)$, any valuation that maps x to true satisfies the formula, no matter the value of y and z. By contrast, in a set circuit C' expressing $x \cup (y \times z)$, the captured set contains the assignments $\{x\}$ and $\{y, z\}$ but not, e.g., $\{x, y, z\}$. In particular, C' is a set circuit in d-DNNF, even though C is not a d-DNNF Boolean circuit.*

We show in [ABJM17] that d-SDNNFs, or even the more general class of d-DNNFs having a *compatible order*, can be converted in linear time to set circuits. This conversion is related to the *smoothing* operation on Boolean circuits [Dar01, SdBBA19]. In our setting, the conversion introduces a new kind of gates called *range gates*, intuitively expressing that a "range" of variables in the v-tree can be assigned in an arbitrary way. The conversion must also enforce the additional requirements that we impose on set circuits to make enumeration simpler, e.g., the constraints on the fan-in of gates. We thus show:

Proposition 3.3.4 ([ABJM17], Propositions 3.9 and 4.3)**.** *Given a satisfiable d-SDNNF C with a v-tree T, we can compute in linear time in C and T a set circuit with range gates which is in d-DNNF and captures exactly the set of satisfying assignments of C (up to omitting the empty assignment).*

Proof step 2: Enumeration for set circuits. The second step is to show the enumeration result on set circuits in d-DNNF, namely:

Proposition 3.3.5 ([ABJM17], Theorem 5.4, Propositions 6.3 and 6.5)**.** *Given a set circuit in d-DNNF (possibly with range gates), we can enumerate the assignments that it captures, with linear preprocessing and delay linear in each produced assignment.*

Note that the delay is independent from the size of the input circuit. The basic idea is simply to apply an easy recursive algorithm:

- For a variable gate g, the singleton set $\mathrm{S}(g)$ is trivial to enumerate.
- For a range gate, we can easily enumerate all possible assignments of its variables.

- For a $\times$-gate, we recursively enumerate the assignments captured by the first input, and we combine each assignment with the assignments obtained by recursively enumerating the second input. Thanks to decomposability, the unions in the relational product are disjoint.
- For a $\cup$-gate, we recursively enumerate the assignments captured by each input. Thanks to determinism, the unions are disjoint.

This algorithm runs in output-linear delay if we assume, like Olteanu and Zavodny [OZ15], that the circuit is *normal* in the sense that no $\cup$-gate is an input to a $\cup$-gate. Note that the output-linear delay bound relies on the decomposability and determinism of the set circuit, and also uses the fact that the captured sets are never empty and that the empty assignment is never captured. However, the situation with non-normal circuits is more complicated, as they may have arbitrarily large $\cup$-*components*: these are intuitively connected components of $\cup$-gates along with their non-$\cup$ inputs. The algorithm sketched above intuitively has a linear delay in the depth of $\cup$-components, which is not output-linear because there can be small assignments that are arbitrarily deep.

The easy solution to this problem would be to rewrite the circuit during the preprocessing to make it normal. However, it is unrealistic to do this in linear time, because it amounts to a transitive closure computation. Instead, we solve this problem by preprocessing the $\cup$-components with a lemma that allows us to do the following: given any $\cup$-gate g of the $\cup$-component, enumerate in constant delay all non-$\cup$ gates g' of the component that have a directed path to g. This uses the fact that, thanks to determinism, these directed paths are necessarily unique, i.e., the $\cup$-components are in fact *multitrees*. Here is the formal claim:

Lemma 3.3.6 ([ABJM17], Theorem C.1)**.** *Given a multitree T, we can preprocess it in linear time to answer the following queries: given any node n of the multitree, enumerate in constant delay the leaves to which n has a path.*

This lemma is easy to show if T is a *tree*, by ordering the leaves according to the traversal order, and precomputing the first and last reachable leaf of each node. The proof of this lemma on multitrees uses an ad-hoc data structure: we intuitively rewrite the multitree to annotate each node with one reachable leaf that can be immediately enumerated.

Our enumeration result on set circuits (Proposition 3.3.5) follows directly from the algorithm sketched above, using Lemma 3.3.6 for the case of $\cup$-gates. This establishes our main result (Theorem 3.3.1).

3.4 Efficient Enumeration for MSO Queries on Trees

We now present an application of the enumeration result on circuits presented in the previous section. Specifically, we study the problem of enumeration for MSO queries on trees, under the measure of data complexity. This result is reproved in [ABJM17] by adapting the provenance circuit construction of my PhD work for probabilistic query evaluation [Ama16] (see Section 5.3). We present the result and proof in this section.

Formally, we fix an MSO formula Q with free variables on an alphabet Σ, we are given a Σ-tree T as input, and we want to enumerate the answers of Q on T. In the general case of free second-order variables, the answers describe the interpretation of each of the sets. Equivalently, we represent each answer as a set of *singletons* of the form $\langle Y : n \rangle$, meaning that the tree node n belongs to the interpretation of the free variable Y.

As pointed out in the introduction, an enumeration algorithm for this problem was given by Bagan [Bag06], and then Kazana and Segoufin [KS13] (if the free variables are first-order):

Theorem 3.4.1 ([Bag06, KS13]). *For any fixed MSO query Q, given an input tree T, we can enumerate the answers of Q on T with linear preprocessing in data complexity, and output-linear delay, i.e., delay linear in the size of each produced answer.*

Note that this evaluation result for MSO extends from trees to *bounded-treewidth structures*, following Courcelle's theorem [Cou90], because a tree decomposition for an input structure of bounded treewidth can be computed in linear time [Bod96].

We show in this section how we can reprove Theorem 3.4.1 using the method presented in the previous section, namely, computing a factorized representation of the answers as a circuit from a tractable class. Beyond re-proving Theorem 3.4.1, we explain in the next chapter how this method can yield new results for incremental maintenance.

Specifically, we show that, for any fixed MSO query $Q(\vec{Z})$, given an input tree T, and letting $X = \{\langle Y : n \rangle \mid Y \in \vec{Z}, n \in T\}$ be the set of possible singletons, we can compute a d-SDNNF on variables X that captures the answers of Q on T:

Theorem 3.4.2 ([ABJM17]). *For any fixed MSO query Q, given an input tree T, letting X be the set of variables defined above, we can compute in linear data complexity a d-SDNNF circuit C over X whose satisfying assignments are precisely the answers of Q on T.*

This result is rather easy to prove, as we explain. Once we have this result, the enumeration result for MSO on trees (Theorem 3.4.1) can simply be obtained in a modular way by combining Theorem 3.4.2 with Theorem 3.3.1.

Proof techniques. This result uses the technique of my PhD [Ama16] to compute *d-SDNNF provenance circuits* for MSO queries (see also Section 5.3). Specifically, the technique works by expressing the MSO query as a tree automaton (which has constant data complexity), and augmenting the input tree to add *assignment facts* denoting the possibilities of assigning free variables to tree nodes, i.e., the singletons $\langle Y : n \rangle$. We then perform a kind of product construction between the automaton and tree, where we consider the singleton facts as variables. The resulting circuit contains one gate for every pair of a node n and an automaton state q, intuitively capturing the satisfying assignments corresponding to the choices of assignment facts in the subtree of T rooted at n for which the automaton will reach state q at node n.

The resulting circuit is in d-DNNF if the tree automaton is deterministic (or even unambiguous), which we can enforce again in constant data complexity. Further, the circuit can be shown to be structured, with a v-tree that can be derived from the input tree. This establishes Theorem 3.4.2.

3.5 Efficient Enumeration for Document Spanners

In this section, we present how our enumeration techniques can apply in the context of *document spanners* [FKRV15]. This is a declarative formalism to specify information extraction tasks on textual documents, where the spanners express which tuples of *spans* (i.e., factors of the input document) should be be extracted. This is a collaboration with Pierre Bourhis, Stefan Mengel, and Matthias Niewerth.

One language to express regular spanners are *regex-formulas*, which are simply regular expressions extended with (well-nested) capture variables denoting the factors to extract. The semantics of a regex-formula on an input word is that we have one result for every extension of the input word accepted by the regex-formula, where the extension can add so-called *marker* characters denoting the assignments of the captured variables. The results thus obtained are called *mappings*: each of them maps the variables of the regex-formula to a *span*, i.e., a factor of the word (identified by its endpoints).

Example 3.5.1. *The following is an example of a regex-formula:*

$$\Sigma^* \sqcup x\{y\{[\hat{}\sqcup]^+\} \cdot z\{[\hat{}\sqcup]^+\}\} \sqcup \Sigma^*$$

This regex-formula describes the task of extracting email addresses from a textual document. Here, an email address is simply a sequence of non-space characters, surrounded by two spaces, and containing the at-sign at a position which is not the first or last character. (Note that this example is simplistic, e.g., it does not allow email addresses at the beginning or end of the word.) The variable x captures each email address, whereas the variables y and z capture respectively what is to the left and right of the at-sign in the captured address.

On an input word, the regex-formula will have as many mappings as there are distinct ways to assign the variables x, y, z so that the result satisfies the regular expression. For instance, on the following word:

0	1	2	3	4	5	6	7	8	9	10	11	12
a	␣	b	@	c	␣	d	␣	e	@	f	␣	g

there will be two mappings. In the first mapping, the variable x will be mapped to the factor containing characters 2–4, and the variables y and z will respectively be mapped to characters 2 and 4. In the second mapping, we map x to the characters 9–11 and y and z to character 9 and character 11 respectively. Ordering the variables as x, y, z, the mappings can be written as tuples in the following way (where spans include the left but not the right endpoint):

$$([2, 5\rangle, [2, 3\rangle, [4, 5\rangle), ([8, 11\rangle, [8, 9\rangle, [10, 11\rangle)$$

The notion of document spanners was originally inspired by practical business needs at IBM [Res18], and research in database theory has focused, e.g., on the expressiveness of spanner representations, on the evaluation of relational algebra queries on tables defined by spanners, and on the compilation of such queries directly into the spanner. The foundational work on the theory of document spanners is the article of Fagin et al. [FKRV15]; see also the PhD thesis of Peterfreund [Pet19].

More recently, the question of producing the extraction results has been investigated from the angle of enumeration algorithms, with some works establishing polynomial delay algorithms to enumerate the mappings of spanner representations [MRV18, FKP18]. An algorithm with linear preprocessing and constant delay was then shown by Florenzano et al. [FRU+18]. However, this algorithm only focuses on data complexity, and not on the complexity in the input spanner.

Our research thus investigated how to enumerate the mapping of a regular spanner on a word with linear preprocessing and constant delay in data complexity, while also ensuring tractability as a function of the input spanner. To formally present our enumeration result, we must introduce the formalism in which we represent regular spanners, namely, *variable-set automata* (VAs):

Definition 3.5.2. *Given a set X of* variables*, the corresponding set of* markers *$\mathcal{M}_X$ consists of the* open markers *$\{\vdash_x | \ x \in X\}$ and* close markers *$\{\dashv_x | \ x \in X\}$. A* variable-set automaton *(VA) on an alphabet Σ and set X of variables is simply a finite automaton over the alphabet $\Sigma \cup \mathcal{M}_X$.*

The VA is sequential *if every run of the VA assigns the markers in a legal way. Formally, for each variable x of X, either there is no marker for x in the run, or there is exactly one occurrence of the open marker $\vdash_x$ of x followed at some later point by exactly one occurrence of the close marker $\dashv_x$ of x.*

The set of mappings *produced by a sequential VA A on a word $w \in \Sigma^*$ is defined as follows: for each word w' of the language of A such that removing the markers of $\mathcal{M}_x$ from w' yields w, we have one result μ, called a* mapping*, intuitively indicating where each variable of X was assigned in w'. Formally, the mapping μ is a partial function from X to the spans $\{[i, j\rangle \mid 0 \leq i \leq j \leq |w|\}$ of w. A variable x has no image by μ if no marker for it occurs in w'. Otherwise, x is mapped by μ to the span $\mu(x) = [i, j\rangle$ of w such that the markers $\vdash_x$ and $\dashv_x$ in w' were at positions corresponding to the span $[i, j\rangle$ of w.*

The formalism of VAs is more expressive than regex-formulas, as these can be translated to VAs in polynomial time [MRV18]. Note that, in contrast with regex-formulas, VAs do not need to assign their variables in a well-nested fashion. We also allow different runs of the VA to assign different subsets of the set of variables X, i.e., to produce "incomplete" mappings: this is in line with the work of Maturana et al. [MRV18] and generalizes the notions of the article of Fagin et al. [FKRV15]. Further observe that the same mapping may be obtained by multiple accepting runs, and also with multiple words w' (which differ only in the order of markers).

Thus, the problem that we study is the following: given a sequential VA A and a word w, enumerate all the mappings of A on w. The algorithm of [FRU+18] can solve this problem with linear preprocessing and constant delay in data complexity, but it first converts the VA into an equivalent deterministic representation, incurring an exponential blowup in the VA. We show that we can avoid this blowup by working directly with non-deterministic VAs:

Theorem 3.5.3 ([ABMN21], Theorem 1.1)**.** *There is a constant $c \in \mathbb{N}$ such that, given a sequential VA A and word w, we can enumerate the mappings captured by A on w with preprocessing time $O(|A|^c \times |w|)$ and delay $O(|A|^c)$.*

The precise complexities are stated in [ABMN21], Theorem 1.1. In particular, if the size of the mappings are assumed to be constant, then we can take $c = \omega + 1$ for the preprocessing and $c = 2$ for the delay, where $2 \leq \omega \leq$ is an exponent for Boolean matrix multiplication (the best currently published bound is $\omega < 2.3728596$ [AW21]). Notice that the delay in this result is only constant

in data complexity (i.e., it does not depend on the length of the input word), but it does depend on the automaton (in a polynomial way). This is in contrast with Theorem 3.3.1 in Section 3.3, and Theorem 3.4.1 in Section 3.4, where the output-linear delay does not hide any dependency in the query.

Proof techniques. The proof of this result implicitly uses a circuit representation of the results of the VA on the word, which is computed by a construction similar to the one presented in the previous section (Section 3.4). Intuitively, the circuit can be thought of as a kind of OBDD with disjunctions (i.e., the non-branching analogue of structured DNNF). Note that the disjunctions are not deterministic because the automaton is not. The challenge is to handle these nondeterministic disjunctions while still producing each result only once.

To do this, the recursive enumeration algorithm remembers a *set* of possible states at the current word position, and efficiently *jumps* over parts of the word where it is not possible to assign variables (so as to ensure the constant-delay bound). This uses a precomputed *jump function* on the product of the automaton and the word, intuitively telling us for each state and position what is the next position where it will be possible to assign a variable, and which states will be reachable — amounting to a limited kind of transitive closure computation. Another proof ingredient in [ABMN19a] is the use of *flashlight search* [SM19] as a subroutine, to efficiently enumerate the sets of markers that can be assigned at a given position.

3.6 Efficient Enumeration for Annotation Grammars

This section reviews a more recent research contribution on enumeration algorithms, in which we generalize from regular spanners to a new formalism of *annotation grammars*. This work is part of Martín Muñoz's PhD work, and is a collaboration with Louis Jachiet and Cristian Riveros.

The practical motivation for annotation grammars is to express information extraction tasks that go beyond regular languages, for instance extracting parts of source code files if we assume that the syntax is defined by a known context-free grammar (CFG), or parts of a structured document. An extension of document spanners to CFGs was recently introduced by Peterfreund [Pet21]: she defines *extraction grammars* and shows that, for a class of *unambiguous* extraction grammars, we can enumerate the mappings of the grammar on an input word with quintic preprocessing and constant delay. We show, via our formalism, that the preprocessing time can be improved to cubic, and can be further improved

on some restricted classes of grammars. In particular, we identify a class that enjoys linear preprocessing and is strictly more expressive than regular spanners (Theorem 3.5.3).

Let us define the grammar formalism that we introduce:

Definition 3.6.1. *Fixing an alphabet Σ and an alphabet Ω of* annotations *(intuitively corresponding to variables), an* annotation grammar *G is a context-free grammar G whose set of terminals can be letters of Σ (describing unannotated letters), or pairs of $\Sigma \times \Omega$ (describing annotated letters). The grammar G thus describes a language of $(\Sigma \cup \Sigma \times \Omega)^*$.*

The set of mappings that G captures on an input word w of Σ^* is defined, like in the case of document spanners, by considering the words w' of the language of G such that w is the result of removing the annotations of w', i.e., replacing each annotated letter by its unannotated counterpart. Each such word w' defines a mapping, which is this time a set of singletons $\langle o : i \rangle$ saying that the annotation $o \in \Omega$ was assigned at position $i \in \{1, \ldots, |w|\}$ of w.

Note the difference with VAs (and with extraction grammars [Pet21]): the annotations can be assigned to an arbitrary subset of the letters (analogously to free second-order variables), and they directly annotate the characters (we do not insert additional marker characters). This superficial difference makes our formalism somewhat more general while simplifying the design of our algorithms. In fact, as we show in [AJMR22], our data complexity results also apply to extraction grammars.

Similarly to [Pet21], we focus on annotation grammars that are *unambiguous* in the sense that, on every input word w, every mapping μ is derived at most once, i.e., the word w' that produces μ has only one derivation. Said differently, the annotation grammar is unambiguous when seen as a CFG over the alphabet $\Sigma \cup (\Sigma \times \Omega)$. We can then show that the mappings of an annotation grammar on an input word can be enumerated efficiently:

Theorem 3.6.2 ([AJMR22], Theorem 3)**.** *Given an unambiguous annotation grammar G and word $w \in \Sigma^*$, we can enumerate the mappings of G on w with preprocessing in $O(|w|^3 \times |G|)$ and output-linear delay (independent of G or w).*

We also introduce in [AJMR22] some classes of annotated grammars where the complexity can be improved. We specifically study *rigid* annotation grammars for which, on any unannotated input word w, there is intuitively only one *shape* of derivation tree for all annotations of w, i.e., all derivation trees for all annotated words w' obtained from w have the same skeleton. We show that the problem for such grammars can be solved with preprocessing only quadratic in the word w, similar to the improved bounds on parsing for unambiguous CFGs.

Last, we introduce a class of *pushdown annotators* (PDAnns), following the usual connection between CFGs and pushdown automata. We then show that the preprocessing can be made linear if the input annotation grammar is expressed as a PDAnn which is *profiled-deterministic*, i.e., for any unannotated input word w, all partial runs on annotations of prefixes of w have the same sequence of stack heights.

Proof techniques. The algorithm used to prove Theorem 3.6.2 can be seen as a variant of the usual CYK parsing algorithm which runs in cubic time in the input word. (There are parsing algorithms for general CFGs which achieve a better asymptotic complexity, in particular following the exponent of Boolean matrix multiplication [Val75], but we are not able to match this bound.) Specifically, we rewrite the input grammar to an *arity-two* normal form [LL09]. Remember that CYK then proceeds by dynamic programming: for every factor of the word and every nonterminal N of the grammar, it determines whether N can derive that factor. In our algorithm, instead of propagating Booleans and combining them with Boolean operations, we construct a factorized representation of the sets of partial mappings intuitively derivable from N on every factor.

The operations that we use to compose these sets of annotations, instead of Boolean operations, are the union and product operations of set circuits (Section 3.3). The resulting circuits can be shown to be in d-DNNF form thanks to the unambiguity requirement (establishing determinism), and thanks to the fact that the assigned singletons can be partitioned according to their position in the word (establishing decomposability). Hence, the captured annotations can be enumerated with output-linear delay (Theorem 3.3.1).

To show the improved bounds on rigid grammars, we follow a similar technique, this time using an existing algorithm to solve parsing for unambiguous CFGs in quadratic time (in the unannotated case): rigidity guarantees that there is always at most one parse tree structure to consider. As for pushdown annotators, the profiled-deterministic condition is designed to ensure that, in addition to rigidity, the unique tree structure for parse trees on the word can be computed deterministically, in linear time. Once the tree structure is known, enumeration can be achieved with linear preprocessing and constant delay by reduction to the same problem for *unambiguous visibly pushdown transducers* of [MR22], which can again be equivalently phrased in the vocabulary of MSO query evaluation over trees (Section 3.4).

3.7 Perspectives

We have presented our contributions to enumeration algorithms in the context of data management. They follow the methodology of knowledge compilation: we establish that d-SDNNF circuits enjoy tractable enumeration (Section 3.3, [ABJM17]), and use this result to efficiently enumerate the results of MSO queries on trees (Section 3.4, [ABJM17, ABMN19b]), the output mappings of spanners (Section 3.5, [ABMN21]), and the annotations captured by annotated grammars (Section 3.6, [AJMR22]).

We hope that the development of enumeration algorithms for data management tasks will continue to lead to interesting and challenging theoretical questions both about databases and about enumeration algorithms. One example of a seemingly central problem is the understanding of which UCQs can be tractably enumerated [CK21], which appears to be very challenging already for the case of CQs with self-joins [CS22].

The perspectives presented here follow several broad directions: some of them are specific to my focus on MSO queries (especially on words), and others are more general. One first question is how the enumeration results obtained so far in database theory can be understood through the lens of factorized circuit representations. Second, we will see how the perspective of practical applications raises several new questions for theoretical research on enumeration. A third question is to generalize enumeration results about document spanners to more expressive languages and automata models. Last, I present some ongoing research directions which explore changes to the enumeration model itself, to allow factorized inputs or to produce outputs by dynamic edits.

Understanding enumeration results through knowledge compilation. This chapter has presented a modular methodology for the design of enumeration algorithms: construct factorized representations of the results to enumerate, argue that they belong to tractable circuit classes, and leverage enumeration algorithms for this class. We introduced the formalism in 2017 [ABJM17], and it has had some echo in the work of other authors since then: it can be used as a technique to show upper bounds, or as a setting in which it is easier to show lower bounds. See for instance the work of Toruńczyk on aggregate queries on bounded expansion databases [Tor20], the work of Berkholz and Vinall-Smeeth on succinct representations of homomorphisms between structures [BVS22], or the work of Muñoz and Riveros on enumeration for visibly pushdown transducers [MR22] and on SLP-compressed documents [MR23] (presented in the very related terminology of enumerable compact sets).

However, there are of course many works in which enumeration results are obtained by seemingly different means. There are also general techniques

for enumeration that have no direct rephrasing as circuits, e.g., the *cheater's lemma* (Lemma 7 of [CK21]), the similar lemma of Durand and Strozecki which assumes efficient testing (Proposition 8 of [DS11]), the technique of *flashlight search* [SM19], etc. Thus, a general research question is to look for a unifying perspective behind the recent wealth of enumeration algorithms over data, in particular in terms of factorized representations and circuits.

Reinterpreting proofs in terms of circuit representations can be useful for theoretical research, to make proofs more modular and more reusable. Further, it could also hold promise for practical implementation. Indeed, it divides the work between query evaluation (extended with circuit computation) and between enumeration (on circuit representations). The first task can hopefully be accomplished by modifying existing database engines, as explored by the ProvSQL system of Senellart et al. [SJMR18]; the second task is related to the development of efficient SAT solvers. Crucially, this modular approach would allow both tools to be developed separately and optimized in an agnostic fashion.

Practical perspectives. We have studied enumeration algorithms with the goal of obtaining efficient theoretical bounds. However, enumerating query answers is of course also relevant in a practical sense: this is obviously the case for the results of database queries, but is also true for the results of document spanners (Section 3.5) when using them for information extraction.

To explore this, I supervised Rémi Dupré on a master's internship aimed at implementing our enumeration algorithm for nondeterministic document spanners (Section 3.5). The project was continued by Matthias Niewerth, and we published experimental results about this implementation [ABMN21]. However, it proved difficult to find or create benchmarks or datasets for the task of enumerating all mappings of a spanner. One promising direction that has been explored since then [BGQ+22] is to connect spanners to the area of *complex event recognition*, which studies how to finding the occurrences of events patterns in a stream of data. Benchmarks for this task are available, e.g., as part of the Grand Challenge[1] research competition of the ACM DEBS conference.

Beyond applying existing algorithms, the study of practical use cases often suggests new directions that theoretical research on enumeration algorithms can then explore. One such central question is that of *ranked enumeration*, i.e., enumerating results in a certain *order* which is relevant to the user. This problem has been investigated recently: by Deep and Koutris [DK21], Tziavelis et al. [TAG+20], by Carmeli et al. [CTG+21] for CQs, and by Bourhis et al. [BGJR21] for MSO queries. Other problems include the computation of *top-k*

[1] https://debs.org/grand-challenges/

results [BDD+22], or performing *ranked access* to results [CTG+21, CZB+22], which subsumes in particular the task of *uniformly sampling* the query results.

In connection to this, an interesting question would be to produce samples of results that are both *diverse* and *representative*, in the sense that they would give a good approximation of the space of possible solutions. However, it is already a challenge to find the correct way to pose this problem and to formally define the goal. A more radical idea is to generalize the notion of enumeration to allow the system to present factorized representations directly to the user, as they could sometimes be more intelligible than a long stream of similar results; however, this also raises questions of user interface design for data exploration.

Enumeration for more expressive languages. One other natural direction for the development of enumeration algorithms is to extend them to more expressive query languages. We present some possible directions for investigation, in the context of enumeration over words. We have first looked at regular spanners (Section 3.5), and then annotation grammars (Section 3.6): a natural direction for further research is to study *enumeration for core spanners*, i.e., spanners who can express equalities between captured spans, or the better-behaved *refl-spanners* [SS21b].

A special case of the enumeration problem for core spanners is to enumerate the matches of so-called *patterns with variables* [MS19], which are strings involving letters and (possibly repeated) variables; for instance, the pattern xx matches square factors of the input word. This pattern language has been studied in the field of *stringology*, which focuses on efficient string matching algorithms; but the question of efficiently enumerating all matches does not seem to have been investigated. Enumeration for core spanners and refl-spanners also seems related to the problem of pattern matching for permutations [BBL98].

Alternatively, if we aim for tractability also in the query, we do not know whether our results that achieve tractability in the automaton (Section 3.5) can also extend to more general automaton representations, e.g., two-way automata, alternating automata, automata with counters, etc. This also raises the question of which circuit classes would correspond to these various classes of automata, a connection which we believe would also be worth investigating.

One last intriguing question is that of enumerating the mappings of document spanners in a *unique* fashion, i.e., enumerating each *tuple of strings* only once. Indeed, enumeration algorithms for document spanners produce each tuple of *spans* only once, but tuples achieving the same factors will be repeated as many times as they occur. In many information extraction settings, however, what matters is to obtain all tuples of strings (e.g., all email addresses), and not all tuples of spans (e.g., all occurrences of the same email address). To my

knowledge, however, this problem does not seem to have been studied.

Other models for enumeration. One last direction for investigation is to explore alternative definitions of enumeration problems, compared to the usual model used in this chapter.

One first idea is to modify the *representation of the input*, for instance to apply to strings that are *compressed* in some way. This was recently studied by Schmid and Schweikardt [SS21a] on SLP-compressed documents, i.e., context-free grammars that concisely express one single string. Their algorithm was then improved by Muñoz and Riveros [MR23]. This follows a general line of research on the extension of string algorithms to SLP-compressed strings [Loh12], and hopefully brings us closer to the same problems for more practical compression algorithms [FT98]. Another way to see these results is that they apply directly to a factorized input: this is more compositional given that many enumeration algorithms are producing factorized representations (implicitly or explicitly).

A second possible twist on the enumeration model, which we are investigating with Mikaël Monet [AM23], is to change what it means to *produce* an output: instead of writing it from scratch (which necessarily takes output-linear time), we can produce it more efficiently by *editing* the previous output. In our work [AM23], we study what are the regular languages whose words can be efficiently enumerated in that sense, by applying edit operations to go from one word to the next (in particular bounding the edit distance between consecutive words). This study could be pursued for other tasks, e.g., enumerating the satisfying assignments of circuits.

This notion of producing new outputs from old outputs is also intriguing in other ways. It is related to factorized representations, given that it can be seen as compressing the output set of results by merging common parts. Further, it also seems linked to algorithms on dynamic data (presented in the next chapter), as it produces the results by performing incremental modifications.

CHAPTER 4

Maintaining Query Results over Dynamic Data

In this chapter, we move on to the study of *incremental maintenance* of query results over dynamic data. The topic has some overlap with the previous chapter, as there has been significant study of incremental maintenance algorithms for enumeration problems, as we will review. I first present the general context, again focusing on data management applications, and present my contributions and perspectives for further research.

4.1 Introduction

Dynamic data is data which may be changed by applying *updates*. When designing algorithms over such data, we must not only compute the output on the current state of the input, but also keep it up to date whenever the data is modified. The goal is to do so more efficiently than recomputing the results from scratch after every update.

Dynamic data is a ubiquitous topic in computer science, in particular in the design of efficient data structures, which are typically mutable, e.g., can store modifiable sets or sequences of elements. Beyond data structures, the theoretical study of efficient algorithms on dynamic data has been studied in several communities, in particular on *graphs* [Hen18] and on *strings* [GFB94, ABR00, GKK+18]; these results have also had some practical impact beyond theory [HHS22, Pre17].

In addition to efficient algorithms, an important topic in theoretical research are *lower bounds*. These come in two main flavors. First, one can prove *uncon-*

ditional bounds, for instance those of Fredman and Saks [FS89] or of Pătraşcu and Demaine [PD06]. Second, following the development of fine-grained complexity [Bri19], we can prove *conditional lower bounds* on problems by reducing from other problems such as Online Boolean Matrix-Vector Multiplication (OMv) [HKNS15], see for instance [AW14].

The question of dynamic data has also been studied in computational complexity through the lens of *descriptive complexity*: if we want to maintain the answer to a problem under updates (along with auxiliary relations), in which *logical language* can we express how the answer and auxiliary relations must be modified after each update? This is the focus of *dynamic complexity* [PI97]; see the survey by Schwentick and Zeume [SZ16]. One landmark achievement of this field is the result by Datta et al. [DKM+18] on reachability in graphs under edge insertions and deletions: it is shown to be in the class DynFO, i.e., can be maintained with changes expressed as first-order formulas [DKM+18]. However, to our understanding, the question of *logical expressiveness* asked by dynamic complexity is rather orthogonal to the question of finding efficient algorithms in terms of *running time*, as indeed the evaluation of FO can take polynomial running time.

In the context of data management, the problem of *maintaining query results* has also been abundantly studied, in particular for the task of *view maintenance* [AHV95, Chapter 22]; see, e.g., [GMS93, CY12]. These results have also inspired practical implementations (e.g., [KAK+14]), and connect to other questions such as machine learning on dynamic data [KNOZ21]. Let us now focus on theoretical results on the incremental maintenance of query answers, following three main lines of research: CQs and UCQs over relational instances, FO over restricted relational instances, and MSO over trees and words.

Maintaining CQs and UCQs. Berkholz, Keppeler, and Schweikardt initiated a study of incremental maintenance for CQs [BKS17] that focused on three problems: maintaining the evaluation of Boolean CQs, maintaining an enumeration structure for the query results, and maintaining the number of results. The database can be changed by inserting and deleting facts. Their work shows (conditional) dichotomies for these problems (assuming self-join-freeness for the case of enumeration). The criterion for the tractability of queries is *q-hierarchicality*, a restriction of the acyclic free-connex criterion which guaranteed tractability in the setting without updates [BDG07]. In a similar vein, more practically-oriented work by Idris et al [IUV17] has tried to adapt Yannakakis's algorithm [Yan81] to the dynamic setting.

The results on CQs in [BKS17] were later extended to UCQs by the same authors [BKS18b]. Their work focuses on the same three problems, along with

that of maintaining a *tuple testing* data structure that can efficiently test whether any candidate tuple is a query answer. However, the lower bound result on enumeration given in [BKS18b] was later found to be incorrect already in the static setting [CK21, Example 2].

Following these dichotomy results, further work has tried to characterize what were the best achievable bounds for the maintenance of the queries classified as intractable. For instance, Kara et al. have studied how to maintain counts for the triangle query [KNN+19]: they investigate the possible trade-offs between the time taken to recompute the query and the space used to materialize intermediate results for faster recomputation; their work is also relevant in the setting of dynamic graph algorithms [HHH21]. Further, some of the authors of [KNN+19] pursued a similar methodology in [KNOZ20]: they investigate the possible trade-offs to efficiently maintain enumeration structures for hierarchical queries (not just q-hierarchical queries). More recent work by the same authors [KNOZ22] studies the maintenance of CQs that are additionally annotated with *access patterns* specifying how the relations can be accessed.

Maintaining FO queries. Going beyond CQs and UCQs, Berkholz, Keppeler, and Schweikardt also investigated incremental maintenance for FO queries with modulo counting quantifiers (FO+MOD). They showed [BKS18a] that updates could be handled in constant time for the three problems mentioned above: maintaining a constant-delay enumeration structure for the results, maintaining the number of query results, and maintaining a constant-time tuple testing structure. Their work assumes that, throughout the edits, the degree of the underlying database remains bounded by a constant.

It is natural to ask whether these incremental maintenance results could be extended to more expressive instance classes on which FO model checking and enumeration are known to be tractable, e.g., nowhere dense graphs. However, this question remains open as of 2022 [SSV22].

Maintaining MSO queries. The incremental maintenance problem can also be studied for MSO queries over trees. This was first investigated in the setting of maintaining *Boolean MSO queries*, i.e., the dynamic membership problem of maintaining whether a tree belongs to a fixed regular language. For this problem, Balmin et al. [BPV04] showed an algorithm achieving $O(\log^2 n)$ time per update, where n is the size of the tree; and achieving $O(\log n)$ time per update for the same problem over words. The dynamic membership problem on words admits in fact a finer $O(\log n / \log \log n)$ upper bound when we only allow substitution updates; see [SFMS97] and also [PI97]. This upper bound

has a matching $\Omega(\log n/\log\log n)$ unconditional lower bound, at least for some Boolean MSO queries [SFMS97]: we review this in more detail in Section 4.4.

Returning to MSO queries over trees, the $O(\log^2 n)$ upper bound of [BPV04] has no known matching lower bound: the results on the *marked ancestor problem* of [AHR98], reviewed in [ABMN19b], also amount to a lower bound of $\Omega(\log n/\log\log n)$ on this problem. It appears that this complexity gap between the upper and lower bound, which already exists for Boolean queries, had not been investigated before our own work on enumeration [ABM18]; see Section 4.3.

The incremental maintenance problem for Boolean MSO queries was later extended to the problem of maintaining *enumeration structures* for the answers of non-Boolean MSO queries. This problem was first studied by Losemann and Martens [LM14]. Their article showed that the update complexity of $O(\log^2 n)$ from [BPV04] sufficed to maintain an enumeration data structure, but their enumeration delay was worse than in the static case: it was no longer constant but $O(\log^2 n)$. These bounds were then refined in further work. For MSO queries on words, Niewerth and Segoufin [NS18] showed an algorithm achieving constant-delay enumeration and $O(\log n)$ update time. For MSO queries on trees, an $O(\log n)$ algorithm with $O(\log n)$ delay on trees was claimed in Niewerth [Nie18], though the proof was later found to be incorrect. We discuss this in more detail, along with our own contributions, in Section 4.3.

4.2 Structure of the Chapter

This chapter presents my contributions on the task of incremental maintenance, which all focus on MSO queries.

In the first section (Section 4.3), I present results on the incremental maintenance of enumeration structures for MSO queries on trees. This extends the results on enumeration presented in the previous chapter (Section 3.4), by allowing the underlying tree to be modified. As it turns out, our circuit-based approach for enumeration is also well-suited to the support of dynamic data, yielding new bounds on the incremental maintenance of this problem.

In the second section (Section 4.4), I present results on the incremental maintenance of Boolean MSO queries on words, i.e., the *dynamic membership* problem of a word to a fixed regular language, under substitution updates to the word. We refine the known $O(\log n/\log\log n)$ upper bound [SFMS97] by investigating how the complexity of this problem depends on the specific regular language that we consider. Improving on earlier work [SFMS97], we identify three main complexity regimes: the complexity is in $\Theta(\log n/\log\log n)$ for some languages where the lower bound of [SFMS97] applies; there are languages

where the problem is in $O(1)$; and there is an intermediate class of languages with an $O(\log \log n)$ upper bound and a conditional lower bound.

4.3 Incremental Maintenance of Enumeration Structures for MSO Queries on Trees

This section presents our result on maintaining enumeration structures for MSO queries over trees. This work extends our results presented in the previous chapter (Section 3.4), and they are joint work with Pierre Bourhis, Stefan Mengel, and Matthias Niewerth.

In the dynamic setting that we consider, we fix an MSO query and evaluate it over an input tree that can be modified by *updates*. We consider *relabeling updates*, where we change the label in Σ of a node to a different value; *deletion updates* where we remove a leaf; and *insertion* updates where we add a new leaf as the first child of a given node or as the next sibling of a given node. When allowing such updates, it is more natural to assume that trees are no longer binary but *unranked*, i.e., each node has an ordered collection of an unbounded number of children. We can easily extend the definitions of MSO evaluation to work on such trees, and tree automaton formalisms can also be adapted.

Whenever the tree is modified, the results of the query on the tree may change. If we want to enumerate the new query results, the naive solution would be to re-run the enumeration algorithm from scratch on the new tree, incurring a linear cost in the tree after each update because of the preprocessing. Our goal is to improve this bound so that the enumeration data structure can be maintained more efficiently after an update is performed. (Note that, however, the enumeration of the query results is always restarted from scratch after an update.)

We studied this problem in [ABM18] where we only allowed *relabeling updates* on trees. In this case, we showed that it was possible to achieve an update time of $O(\log n)$ and a delay of $O(1)$, i.e., the same delay as for enumeration on static data, and the same update complexity as for the earlier known results on words [NS18] (but which further allows insertion and deletion updates). Specifically, we show the following strengthening of Theorem 3.4.1:

Theorem 4.3.1. *For any fixed MSO query Q, given an input tree T, we can enumerate the answers of Q on T with linear preprocessing in data complexity and output-linear delay, and we can update the structure after any relabeling update to T in time $O(\log |T|)$.*

Proof techniques. It is easy to prove Theorem 4.3.1 for substitution updates with an update complexity in $O(h)$ where h is the height of the tree (which is unchanged by the substitution operations). To do this, one simply needs to notice that both the construction of the circuit in Theorem 3.4.2, and the precomputation phase in Theorem 3.3.1, can be performed *bottom-up* on the input tree T. Hence, whenever we perform a relabeling update on a node n of the tree T, we only need to recompute the circuit and to re-do the precomputation phase along the path from n to the root. The rest of the enumeration structure can be re-used, yielding an $O(h)$ algorithm.

To achieve $O(\log n)$ complexity, we need to argue that we can work with balanced trees. This is easier under relabeling updates, because the structure of the tree never changes. Specifically, we use for this the somewhat inelegant black-box technique of computing a balanced tree decomposition of constant width of the input tree, which can be done in linear time during the preprocessing by the result of Bodlaender [BH98]. We can then translate the MSO query on the original tree to an MSO query on the balanced tree encoding, and translate relabeling updates from the original tree to the balanced representation. This yields the bound of Theorem 4.3.1.

A stronger result was claimed in [ABMN19b], namely, that the bound of Theorem 4.3.1 also applies if we allow insertion and deletion updates on the tree. Unfortunately, the status of our result is currently unclear, because it depends on a result whose proof was later found to be flawed. Specifically, our technique in [ABMN19b] relies on a *balancing scheme*, where we must maintain a balanced representation of the current tree under insertion and deletion updates, in such a way that MSO queries on the original tree could be equivalently expressed in MSO over the balanced representation. The existence of such a balancing scheme was claimed in Niewerth [Nie18], but the proof of that result was later found to be incomplete. A corrected proof of the same claim was recently posted as a preprint by Kleest-Meißner, Marasus, and Niewerth [KMMN22]; if correct, this will complete the proof of our result in [ABMN19b].

4.4 Dynamic Membership for Regular Languages on Words

In this section, we present our results on the dynamic membership problem for regular languages on words under substitution updates [AJP21]. This means that we move back from the setting of enumeration to that of Boolean MSO queries, we move from trees back to words, and we only allow substitution updates. In exchange for these restrictions, we will aim for a much finer classification of

regular languages, by showing bounds that depend on the language that we fix.

As we explained in the introduction, there is an general upper bound of $O(\log n/\log\log n)$ on this problem for any fixed regular language [SFMS97], where n is the word length; and this bound is matched by an unconditional $\Omega(\log n/\log\log n)$ lower bound for some regular languages [SFMS97], which follows from the results of Fredman and Saks [FS89]. However, the complexity of the problem can be lower, depending on the regular language:

Example 4.4.1. *Consider the language a^* on the alphabet $\Sigma = \{a, b\}$. The membership of a word to this language can be maintained in constant time under substitution updates. To do this, we simply maintain a counter of the number of occurrences of each letter in the word, that we increment or decrement after each update. The current word is in a^* iff the current number of b's is zero. (Remark how this scheme uses our ability to perform arithmetic operations in constant time per operation; see [SFMS97, PT07a] for results in a different model.)*

Our goal is thus to characterize what is the complexity of dynamic membership depending on the regular language. We identify three complexity regimes. For some languages, the complexity is $\Theta(\log n/\log\log n)$, as follows from the upper and lower bounds of [SFMS97]. For *all* languages not covered by the lower bound, we show an $O(\log\log n)$ algorithm, generalizing the algorithm for star-free languages given in [SFMS97]. Further, we show an $O(1)$ algorithm for a subclass of these languages. The $O(1)$ class contains in particular the commutative languages like in Example 4.4.1 (their tractability was already noticed in [SFMS97]); but it also contains some more surprising cases, e.g., the language $c^*ac^*bc^*$.

When the $O(1)$ algorithm does not apply, we can show a conditional lower bound based on a problem called *prefix-U_1*. This problem asks us to maintain a word over the alphabet $\{0, 1\}$ under substitution updates, and to answer queries asking, given a prefix length i, whether the prefix contains some occurrence of the letter 0. Equivalently, we must maintain a set of integers under insertions and deletions, and be able to compare the current minimum of the set to any input integer, all in constant time. We can show that this problem reduces to the dynamic membership problem for any language covered by the $O(\log\log n)$ algorithm but not the $O(1)$ algorithm. Hence, if we assume that *prefix-U_1* cannot be solved in constant time on the RAM model, then the same is true of the dynamic membership problem for each of these languages.

Here is the formal statement of our result, for some regular language classes **QLZG** and **QSG** to be defined later:

Theorem 4.4.2 ([AJP21]). *Let L be a fixed regular language, and consider the problem of maintaining membership to L on a word of length n under substitution updates:*

- *If L is in **QLZG**, then the problem is in $O(1)$ per update.*
- *If L is in **QSG** but not in **QLZG**, then the problem is in $O(\log \log n)$ per update. Further, solving the problem in time $O(1)$ per update gives a data structure for the prefix-U_1 problem in $O(1)$ time per operation.*
- *If L is not in **QSG**, then the problem in in $\Theta(\log n / \log \log n)$ per update.*

Defining the regular language classes. The definition of the classes **QLZG** and **QSG** are unfortunately somewhat technical, as they use algebraic notions on regular languages. To define them, we first introduce the *syntactic monoid* of a regular language L:

Definition 4.4.3. *The* syntactic equivalence relation *of a language L is the relation on Σ^* defined as follows: two words u and v are syntactically equivalent if, for any prefix s and suffix t, the word sut is in L if and only if the word svt is in L. The* syntactic monoid *M of L is the quotient of Σ^* by the syntactic equivalence relation.*

Informally, syntactic equivalence means that two words "behave the same" and can be substituted for one another when we are interested about membership to the language L. The syntactic monoid is built on the equivalence classes of this syntactic equivalence relation, with a law that corresponds to the concatenation of words, and a neutral element corresponding to the empty word. It is well-known that, as the language L is regular, it has only finitely many syntactic equivalence classes, and thus its syntactic monoid is finite. We let η be the *syntactic morphism*, i.e., the function mapping each word of Σ^* to the element in M that corresponds to its equivalence class, and we consider the set of equivalence classes corresponding to single letters, i.e., $\Lambda := \{\eta(a) \mid a \in \Sigma\}$.

We can now define the *stability index* of L as the idempotent power of Λ in the so-called *powerset monoid* of the syntactic monoid M. Formally:

Definition 4.4.4. *Let M' be the* powerset monoid *of the syntactic monoid M: its elements are the subsets of M, its composition law is defined by $EF := \{xy \mid x \in E, y \in F\}$ for $E, F \subseteq M$, and its neutral element is $\{e\}$ for $e = \eta(\epsilon)$ the neutral element of M. Note that M' is a finite monoid, and that Λ is an element of M'. The* stability index *of the language L, denoted ι, is then the* idempotent power *of Λ in M', i.e., the least positive integer ι such that $\Lambda^{2\iota} = \Lambda^{\iota}$.*

Intuitively, the element Λ of the powerset monoid is the set of equivalence classes of M that can be achieved by single letters; the element Λ^2 is the set of equivalence classes that can be achieved by words of exactly two letters; and so on. The stability index ι is then the idempotent power of Λ, satisfying $\Lambda^\iota = \Lambda^{2\iota}$ so that in fact $\Lambda^\iota = \Lambda^{n\iota}$ for any $n > 0$. This means that the stability index has the following key property: any element of the syntactic monoid that can be achieved by a word of length $n\iota$ for some $n > 0$ can also be achieved with a word of length ι. This intuitively means that, when considering membership to the language L, the stability index ι is the smallest size of "blocks" of contiguous letters such that the "effect" of any sequence of blocks can also be achieved by a single block (and can thus be realized by a substitution update).

To formalize this intuition, we define the *stable semigroup* of the language L as the subsemigroup of the syntactic monoid M generated by the syntactic classes of the words of length ι, i.e., its set of elements is $\Lambda^\iota = \{\eta(w) \mid w \in \Sigma^\iota\}$. Intuitively, these elements are the equivalence classes of the syntactic equivalence relation that can be achieved by blocks of size ι. Note that the stable semigroup is indeed a subsemigroup of M, because the definition ensures that the composition of two such classes can also be achieved with a single block, hence it also belongs to the stable semigroup. However, the stable semigroup is in general not a submonoid: it may be the case that the equivalence class $\eta(\epsilon)$ of the empty word cannot be achieved with a block.

We can finally define the class **QSG** of regular languages for which we give an $O(\log \log n)$ algorithm: it is the class of languages whose stable semigroup is in the class **SG** of semigroups defined by the equation $x^{\omega+1}yx^\omega = x^\omega yx^{\omega+1}$, where x and y range over the elements of the semigroup and ω denotes the idempotent power of elements. This class of semigroups, where **SG** means "swappable groups", had only been mentioned incidentally in [dA90]. The equation intuitively means that elements belonging to the same subgroup of the semigroup can be "swapped".

The definition of the class **QLZG**, for which we give an $O(1)$ algorithm, is slightly more involved. It is the class of regular languages whose stable semigroup is in the class **LZG** of semigroups defined by requiring that all submonoids are in a class **ZG**. The class **ZG** of monoids, introduced in [Aui00], is in turn defined by the equation $x^{\omega+1}y = yx^{\omega+1}$. This equation, a stronger requirement than the **SG** equation, means that elements belonging to a subgroup of the monoid are *central*, i.e., they commute with all other elements; hence the same "zentral group". To summarize, **QLZG** is the class of regular languages where, in all submonoids of the stable semigroup, all subgroup elements are central.

Proof techniques: Monoids. The proof of Theorem 4.4.2 uses notions from algebraic monoid theory [Pin86, Pin19] combined with algorithmic tools. In particular, to show the results on regular languages, we first consider the analogous problem on monoids, and then on semigroups. Formally, this is the *dynamic word problem*: we are given a word w of elements of a monoid or semigroup, and must maintain the evaluation of w according to the monoid or semigroup law, under substitutions on the elements of w, and measuring the worst-case complexity per update as a function of the length of w.

The dynamic word problem for monoids had been studied by Frandsen et al. [SFMS97]. One first observation is that the complexity of the problem is unchanged when taking a submonoid of a monoid, a quotient of a monoid, or the direct product of two monoids. Hence, for any asymptotic complexity regime, the class of monoids that enjoy this complexity for the dynamic word problem forms a *variety*, i.e., is closed under these operations.

To identify for which monoids the dynamic word problem is in $O(1)$, we start with the result of [SFMS97] which shows this for commutative monoids. For such monoids, we can simply count the number of occurrences, as in Example 4.4.1. We show that we can also achieve $O(1)$ complexity for another class of monoids called **MNil**, formally the monoids obtained by adding a neutral element to a *nilpotent semigroup*, covering for instance the language $c^*ac^*bc^*$. These monoids are not commutative, but because of nilpotency, we can only combine a constant number of non-neutral elements before the result is zero. We show that the dynamic maintenance problem for such monoids can be solved in $O(1)$, by storing the occurrences of non-neutral elements in unordered doubly linked lists. We then show [AP21] that the class **ZG** defined above can be equivalently characterized as the variety generated by **MNil** and by commutative monoids, which implies that the dynamic word problem is in $O(1)$ for such monoids.

Outside of **ZG**, we show that the dynamic word problem is in $O(\log \log n)$ for the monoids which satisfy the equation of **SG**. This includes aperiodic monoids, for which this was already known [SFMS97], but also some other monoids. We use the standard technique of doing an induction on $\mathcal{J}$-classes of the monoid, and inspired by [SFMS97] we use a van Emde Boas tree [vEBKZ76] data structure to answer *predecessor*/*successor* queries in $O(\log \log n)$ and "jump over" the elements not in the class of interest. We also use the Rees-Sushkevitch theorem [Pin19] to understand the composition law on each $\mathcal{J}$-class.

As for monoids outside of **SG**, they happen to be precisely the ones covered by the $\Omega(\log n / \log \log n)$ lower bound of [SFMS97], as can be shown from the equation of **SG**. Similarly, for **SG** monoids which are not in **ZG**, the choice of elements violating the equation of **ZG** can be used to design a reduction from the prefix-U_1 problem.

Proof techniques: Semigroups and languages. Having shown these results on the complexity of the dynamic word problem for monoids, we first lift them to semigroups, and then to dynamic membership for regular languages. For the dynamic word problem on semigroups, from the classes **SG** and **ZG** of monoids, we consider the class **LSG** of semigroups where all submonoids are in **SG**, and the class **LZG** of semigroups where all submonoids are in **ZG**. Clearly, for semigroups not in these classes, we can show complexity lower bounds by considering only the submonoids that are not in the prescribed class; so the challenging part is to show upper bounds.

For **SG**, this turns out to be easy, because the class **LSG** of semigroups turns out to be precisely those satisfying the equation of **SG** (as semigroups), and our upper bound directly applies to them. For **ZG**, this is more complicated, because there are semigroups in **LZG** who do not satisfy the equation of **ZG**. To cover these, we first show that our $O(1)$ upper bound on **ZG** can in fact be extended to the variety $\mathbf{ZG} * \mathbf{D}$ generated by semigroups that combine a **ZG** monoid with a semigroup in the class **D** of *definite* semigroups via a certain *semidirect product* operation. We then establish in a separate paper [AP21] that this class $\mathbf{ZG} * \mathbf{D}$ is in fact equal to **LZG**. This is a so-called *locality* result, proved via Straubing's delay theorem on the category of idempotents [Str85].

Last, we lift the results from semigroups to regular languages. This is easily done using the notion of the stable semigroup introduced earlier. Intuitively, we can group letters into "blocks" without worsening the complexity, because we can compute moduli and divisions in constant time in the RAM model. Thus, dynamic membership to a regular language can be reduced to the dynamic word problem for the stable semigroup of that language. Conversely, lower bounds on the latter problem can be lifted to dynamic membership — this uses the fact that all elements of the stable semigroup can be achieved by blocks of the same size.

4.5 Perspectives

This chapter has presented my research on incremental maintenance for data management, and introduced our contributions on the maintenance of MSO queries. We have approached this problem from two angles: enumeration structures on trees, and more precise complexity bounds for Boolean queries on words.

We believe that these results illustrate that there is much more to understand about the problem of query evaluation on dynamic data, and that it leads to questions both algebraic and algorithmic, bridging areas such as formal language theory and data structures. We now give examples of concrete directions in

which this investigation can be pursued, as well as more open-ended questions about incremental maintenance.

Extending to richer languages. One clear direction for future research is to extend the results of Section 4.4 from regular languages to more general languages. The most immediate direction is to extend to *tree languages*, i.e., the maintenance of regular languages over trees under relabelings. This is a problem for which there is a $O(\log n)$ upper bound (as we showed in Section 4.3) and $\Omega(\log n / \log \log n)$ lower bounds (for the marked ancestor problem [AHR98], and for regular word languages [SFMS97]); but of course there are also some languages enjoying a better complexity. In contrast with the setting of words, obtaining definitive results for the problem on trees may be out of reach, because the algebraic theory of tree languages is not as well developed as that of word languages. Nevertheless, we can hope that the complexity can be characterized at least for some classes of languages.

A similar direction, looking at languages on words, is to investigate how the algorithms for dynamic membership can be extended to support more expressive languages, e.g., context-free languages. In this setting, some initial results were obtained for specific languages in the nineties [FHM+95, HR98]; it may be possible to extend results using more recent tools. The dynamic membership problem for context-free languages also seems related to the wider area of *incremental parsing* [WG98], which studies how to maintain a parse tree for a string under updates; it can also be relevant in practice, e.g., to maintain the structure of code as it is edited, for instance within an Integrated Development Environment (IDE).

More ambitiously, one can ask whether some of our techniques can be used in the wider context of maintaining properties of graphs, e.g., of bounded-degree graphs in the spirit of [BKS18a]. The latter work shows that we can maintain query results (indeed, even constant-delay enumeration structures) with constant time per update for first-order logic on such structures; but we may be able to extend this result to more general languages using some of our methods in [AJP21].

Establishing lower bounds. Another general direction for research on incremental maintenance is to show more lower bounds, already in the context of dynamic membership for regular languages on words (Section 4.4). In particular, we only conjecture that the problem prefix-U_1 can not be solved in constant time. Proving this is a natural open question, but probably a challenging one. Likewise, some of the $O(\log \log n)$ languages can be shown to have a matching lower bound in $\Omega(\log \log n)$, using known results on the so-called *colored*

predecessor problem [PT07b]. The impact of this is unclear, in particular we do not know what is the subclass of the languages in **QSG** \ **QLZG** for which the complexity is in $\Theta(\log \log n)$. We expect that this would not be the case of all languages, see [AJP21].

Supporting more expressive edit operations. Our work on incremental maintenance has focused on *substitution updates*. A natural question is whether our results on dynamic membership for regular languages (Section 4.4) can be extended to also support insertion and deletion operations. This is not directly possible because of an $\Omega(\log n / \log \log n)$ lower bound (see [Jac]) on the problem of incremental maintenance of a word under insertion and deletion updates ("insert/delete a character before/after position i") and very simple queries ("retrieve which character is at position i"). However, we may be able to support insertions and deletions when assuming that the word is stored in a doubly-linked list, with update operations given as explicit pointers to the affected nodes. It may be possible to show results in this setting by adapting our result on substitution updates, leveraging known data structures for the *order-maintenance problem* [BFG+17].

Another question concerns the more restricted case of insertions and deletions that can only be done at the beginning and end of the word, i.e., push and pop updates. This question is related to *sliding window* algorithms, which read text in streaming (by adding/removing characters at the ends of the window) and must maintain information about the word currently in the window. For instance, recent work by Ganardi, Jachiet, Lohrey, and Schwentick [GJLS22] has shown (among other things) that dynamic membership to a regular language can be maintained in constant time under such updates. It is not clear whether this result can be combined, for instance, with substitution updates.

More ambitiously, it would be interesting to study the effect of complex updates that affect words or trees in a more widespread way. For instance, we could study the effect of cut-and-paste operations, i.e., of *splitting* and *joining* strings, or trees; we are not aware of complexity bounds under such operations. Another kind of expressive update operations are those changing several characters at a time, e.g., search-and-replace. A related notion of *bulk updates* has been studied in the setting of dynamic complexity, for instance in Schwentick, Vortmeier and Zeume [SVZ18], who study changes defined as first-order queries. It would be interesting to study similar problems under the angle of efficient algorithms or lower bounds on the running time.

Update complexity beyond Boolean queries. One last natural direction for future investigation is to maintain the result of more expressive queries. Our

contributions have focused on maintaining the answers of Boolean queries (Section 4.4), and maintaining enumeration structures (Section 4.3). In fact, for the results of Section 4.4 on Boolean queries, it is already a natural question whether the results can extend to, e.g., the maintenance of enumeration structures with update complexity below $O(\log n)$ for some languages.

However, beyond enumeration structures, there are many other informations that one can wish to maintain. We mentioned in the introduction the problems studied in [BKS18a, BKS18b]: maintaining the *number* of results, and maintaining efficient *tuple testing* structures that can check if a candidate solution is indeed an answer. These problems had in fact already been studied in the static setting (i.e., without updates): for FO queries on databases with local bounded expansion [SV17], for FO queries on nowhere dense databases (for counting [GS18] and testing [SSV22]), and for MSO queries over trees [Kaz13, KS13]. An intriguing question is which of these results can be extended to the dynamic setting, beyond the known results [BKS18a, BKS18b, KNN+19] and what are the possible trade-offs, e.g., keeping in mind the lower bounds on the marked ancestor problem [AHR98] for MSO queries over trees.

There is also a specific connection between incremental maintenance and tuple testing, in the sense that testing can sometimes be implemented by modifying the query and by performing updates that describe the tuple to be tested. We use this connection in [AJP21] to show bounds on prefix, suffix, and infix queries on dynamic data. In generally, however, it is unclear whether there is a unifying approach that can simultaneously give algorithms for the tuple testing problem and for the problem of incremental maintenance.

Further, on words and trees, in addition to counting and testing, there are other kinds of natural information that one can maintain: the *number of accepting runs* of a nondeterministic automaton, the *probability* of the current word according to a weighted automaton, the smallest *edit distance* of the current word to a word of the language [WG98], etc. All of these are promising directions to investigate.

CHAPTER 5

Query Evaluation over Probabilistic Data

This last chapter presents the area of query evaluation on *probabilistic data*. This task has a conceptual connection to incremental maintenance, because we are in a sense performing weighted counting over all possible modified versions of the input data; but the techniques are in fact quite different. The chapter is again structured as an introduction to the area, followed by the presentation of my contributions and some directions for future research.

5.1 Introduction

When designing algorithms, there are many settings in which we do not know what is the true state of the input data. There can be uncertainty over whether individual data items are correct or not; or we may want to anticipate the risk that some items no longer hold and estimate how much the computation result is affected by these possible errors. These notions of uncertainty are a very general topic that has been studied from the angle of logics and reasoning [Hal17], e.g., via fuzzy set theory, possibility theory, Dempster-Shafer theory, etc. In the more quantitative approach of probability theory, the question of reasoning with uncertainty has been studied in the setting of graphical models (such as Bayes networks or Markov networks) and the task of probabilistic inference; or in the area of probabilistic programming [DRKT07].

In the area of data management, the question of uncertainty has been immediately motivated by the practical issue of data quality [Fan15]: real-world databases contain inaccuracies, stale data, missing data, integrity constraint

violations, missing values (NULLs), etc. One common approach is to perform data cleaning [RD00, CIKW16] and resolve the uncertainty; or to reason about all possible ways to repair the data, as is done in *consistent query answering* [ABC99, KW21]. The other approach is to directly reason with incomplete information; this was advocated for instance by Imieliński and Lipski [ILJ84], and has lead among other things to a fruitful study of the problem of *open-world query answering* [CLR03].

Among these lines of work, we focus specifically on the study of *probabilistic data* on relational databases, where the relational model is extended to allow facts annotated with probability values: see the book of Suciu et al. [SORK11] which surveys this area. The simplest model for probabilistic relational data is *tuple-independent databases* (TID), which are relational databases where every fact is annotated by a probability value. We assume that each fact is present with the indicated probability, and absent otherwise; all these probabilistic events are further assumed to be independent. The expressiveness of the TID model is rather limited: it does not make it possible, e.g., to express mutually exclusive values or incompatibilities between facts (as can be done, e.g., in the *block-independent disjoint* formalism [BGMP92, RS07]), uncertainty on data values, correlations, etc.

Yet, the TID model has already proven quite challenging to understand, in particular when evaluating queries. This task, which we call *probabilistic query evaluation* (PQE), is defined as follows: we fix a Boolean query, we receive a TID instance as input, and we must determine the probability that the query is true, namely, the total probability of the possible states of the data where the query holds. This problem was initially called *query reliability* and has been investigated from the nineties [dR95, GGH98], inspired by earlier work on the *reliability problem* in networks [Col91, Val79].

Probabilistic query evaluation can be solved by nondeterministically guessing a possible state of the data (biased by its probability), checking whether the query holds (which we assume can be done in polynomial data complexity), and summing over all nondeterministic draws. This means that, up to the polynomial-time computation needed to normalize the probability, the PQE problem belongs to the class #P of counting problems that can be expressed as counting the number of accepting runs of a nondeterministic polynomial-time Turing machine. The question is then to understand if one can design better algorithms for PQE, e.g., solve the problem in (deterministic) polynomial time.

Initial results on the complexity of PQE [GGH98] have shown that the problem was *#P-hard* already for some conjunctive queries. By contrast, it is obvious that the problem is in polynomial time for some trivial queries, e.g., $\exists x\ R(x)$. The natural question is then to characterize the queries for which PQE is tractable. This was first accomplished by Dalvi and Suciu [DS07] on the

class of *self-join-free conjunctive queries*: they show that the *hierarchical* queries enjoy tractable PQE, whereas all others are #P-hard. The dichotomy was then extended by Dalvi and Suciu [DS13] to a much more challenging result showing that each UCQ was either *safe* (PQE is in PTIME) or *unsafe* (PQE is #P-hard).

Other works have investigated the complexity of other query languages, e.g., queries that feature negation [FO16], or disequality ($\neq$) joins [OH08], as well as inequality ($<$) joins [OH09]. More recently, other works have investigated, e.g., incremental maintenance for PQE on probabilistic databases [BM21], probabilistic databases on infinite domains [CGLS21], or probabilistic databases with uncertain numerical values [CLP23].

5.2 Structure of the Chapter

This chapter presents my work on the probabilistic query evaluation problem (PQE), following three main research directions.

The first direction (Section 5.3), started during my PhD and pursued afterwards, concerns restrictions on the structure of the input instances. My PhD showed that PQE was tractable for MSO queries when the input instances are required to have bounded treewidth. Otherwise, under some technical assumptions, the problem was shown to be intractable for some queries, in the sense that PQE is computationally hard and also that there are no small tractable provenance representations. With Mikaël Monet, during his PhD and after, we were able to improve on these intractability results, showing that they apply to simpler queries and to more expressive provenance formalisms.

The second direction (Section 5.4), explored with İsmail İlkan Ceylan, also studies the PQE problem for expressive query languages, but this time without restricting the input instances. We show [AC20] that the PQE problem is intractable for *any* query on an arity-two signature which is *homomorphism-closed* and *unbounded* (i.e., not equivalent to a UCQ).

The third direction (Section 5.5), studied with Benny Kimelfeld, concerns the *unweighted* PQE problem, which we call *uniform reliability* (UR). This is the restricted case of PQE where we require all facts of the input TIDs to have a probability of $1/2$. We show a dichotomy for the complexity of UR for CQs without self-joins, based on the same criterion as for PQE [DS07] but with a more involved hardness proof. Recently [Ama23], I have further extended the study of UR to the class of queries studied in [AC20]; I show that the hardness of PQE for such queries already holds for UR.

5.3 Intractability over Unbounded-Treewidth Families

Our first results on probabilistic query evaluation (PQE), presented in this chapter, investigate how the tractability of PQE depends on the structure of input instances, specifically their treewidth. This question was one of the main topics of my PhD [Ama16], where I had shown that bounding instance treewidth makes PQE tractable, and that this was in a sense the only structural restriction on instances that guaranteed tractability.

Specifically, my PhD showed an upper bound [ABS15, ABS16] establishing that PQE was tractable for MSO queries on trees, and on bounded-treewidth data following Courcelle's theorem [Cou90]. This is in line with earlier results on probabilistic XML [CKS09], and with tractability results for probabilistic inference on bounded-treewidth models [LS88, HD96], This upper bound [ABS15, Ama16] was shown using the so-called *intensional approach* to PQE: we first compute the provenance of the query in a tractable representation, specifically structured d-DNNF circuits (d-SDNNFs); and we then use these circuits to tractably compute the probability. Notice that this intensional approach is the analogue of the circuit-based modular approach presented in Chapter 3 for the enumeration of query answers, and is in fact the origin of our techniques in Section 3.4 for this task.

My PhD further gave lower bounds showing that, in a certain sense, the *only* way to guarantee the tractability of PQE for MSO queries is to bound the treewidth of instances. More precisely, in [ABS16, Ama16], with Pierre Bourhis and Pierre Senellart, we showed two main intractability results on unbounded-treewidth data, restricting to arity-two signatures for technical reasons.

The first intractability result is a *lower bound on provenance representations*: we showed that there are connected UCQ$^{\neq}$ queries whose provenance cannot tractably be represented by OBDDs on unbounded-treewidth instances, i.e., the size of an OBDD that expresses the provenance of the query must be exponential in the instance treewidth. This result implies that PQE for these queries cannot be solved with the intensional approach if we use OBDDs as our choice of tractable provenance representation.

The second intractability result is a *hardness result*: if we fix *any* infinite family $\mathcal{I}$ of non-probabilistic arity-two instances whose treewidth is unbounded, and where high-treewidth instances can efficiently be constructed in a certain sense, then we show that PQE cannot be tractable even when restricted to input instances in the family $\mathcal{I}$ (with arbitrary probabilities). More precisely, there are monotone MSO queries, or non-monotone FO queries, for which PQE is intractable on any such family $\mathcal{I}$, specifically it is #P-hard under randomized re-

ductions. The result implies that, under the technical assumptions that we make, bounding the treewidth is the only way to restrict the structure of instances that makes PQE tractable for all FO queries, or for all monotone MSO queries.

After my PhD, and in particular during the PhD thesis of Mikaël Monet, we continued to study the intractability of PQE on high-treewidth families. We were able to improve both of the above results, as I now present.

Lower bound on provenance representations. The first direction was investigated as part of the PhD thesis of Mikaël Monet, together with Pierre Senellart. Our study focuses on the class of connected UCQ$^{\neq}$ queries introduced in [ABS16], called the *intricate queries*. We showed [AMS18] that the lower bound of [ABS16] on provenance representations as OBDDs for intricate queries could be generalized to the more expressive class of d-SDNNF circuits:

Theorem 5.3.1 ([AMS18], Theorem 33)**.** *For some integer $d \in \mathbb{N}$, on any arity-two signature σ, considering any instance I over σ and any connected UCQ$^{\neq}$ Q on σ which is intricate in the sense of [ABS16], then any d-SDNNF representation of the provenance of Q on I must have size $2^{\Omega(k^{1/d})}$, where k is the treewidth of I.*

This means that, for intricate queries on high-treewidth instances, we cannot tractably solve PQE via the intensional approach using d-SDNNF circuits. This contrasts with bounded-treewidth instances, where this approach is tractable [ABS15, Ama16]. However, Theorem 5.3.1 does not necessarily imply that PQE cannot be solved by other approaches. For instance, there are UCQs admitting tractable provenance representations in the class of *d-Ds* (i.e., d-DNNFs without the NNF restriction) but not in the class of d-SDNNFs [BS17, Mon20], and the tractability of the safe UCQs of [DS13] is generally not known to be explained by the intensional approach [Mon20]. This is the motivation for the computational hardness results which we present next.

Hardness of PQE on unbounded-treewidth families. Second, together with Mikaël Monet but after his PhD, we extended the hardness result of PQE on unbounded-treewidth instance families, to apply to queries in less expressive languages than MSO or FO. In fact, we could show [AM22] that the hardness of PQE on such families already held for the class of connected UCQ$^{\neq}$ queries. Our results on an arbitrary arity-two signature σ are shown with the connected UCQ$^{\neq}$ Q_σ below, where we write σ_2 for the set of binary relations of σ:

$$Q_\sigma : \left(\bigvee_{R\in\sigma_2} R(x,y) \vee R(y,x)\right) \wedge \left(\bigvee_{R\in\sigma_2} R(x,z) \vee R(z,x)\right) \wedge x \neq y \wedge x \neq z \wedge y \neq z$$

Note that for brevity we write Q_σ using nested unions, but we can easily rewrite it as a UCQ simply using distributivity. Intuitively, Q_σ asserts that there are two binary facts in the instance which share exactly one element. This query happens to be an intricate query, so it is covered by the lower bound on d-SDNNFs presented above, but our hardness result shows that PQE for this query is also computationally hard on any instance family $\mathcal{I}$ in which high-treewidth instances can be constructed efficiently:

Definition 5.3.2. *A family of instances $\mathcal{I}$ is* treewidth-constructible *if there is a algorithm which, given a number k, computes in polynomial time in the value of k an instance I of $\mathcal{I}$ whose treewidth is at least k.*

Our result is then the following:

Theorem 5.3.3 (Follows from [AM22])**.** *Let $\mathcal{I}$ be a treewidth-constructible family of instances on an arity-two signature σ. The PQE problem for the query Q_σ is #P-hard under randomized (ZPP) reductions.*

Here, ZPP is the class of zero-error probabilistic polynomial time, consisting of randomized algorithms that run in polynomial time but may fail with some constant probability.

Proof techniques. All the results presented in this section rely on the technique of finding large grids as minors of high-treewidth graphs, specifically the polynomial bound on the grid minor theorem shown by Chekuri and Chuzhoy [CC16]; earlier such results had already been used in previous work to show the intractability of logical problems [KT10, GHL+14].

We rephrase these results to the setting of *topological minors*. Specifically, let us consider degree-3 undirected graphs (i.e., every vertex has at most 3 neighbors) which are planar (i.e., that can be embedded on the plane without edge crossings). The result of [CC16] implies that any planar degree-3 graph H with n vertices can be found as a topological minor of any graph G of treewidth at least n^{100}, i.e., some subgraph of G is a subdivision of H. Further, given G and H, we can find such a topological embedding in randomized polynomial time — this use of randomization in [CC16] is why Theorem 5.3.3 uses ZPP reductions.

Thanks to these results, and using treewidth-constructibility, we can follow a simple idea to show the intractability of PQE on a treewidth-constructible family $\mathcal{I}$. First, given any degree-3 planar graph H, we use treewidth-computability to find an instance I in the fixed family $\mathcal{I}$ whose Gaifman graph G has sufficiently high treewidth. Second, we use [CC16] to find a subgraph G' of G which is isomorphic to a subdivision of H, and eliminate the other edges by giving a

probability of 0 to the corresponding facts of I. It now suffices to show how an intractable problem on H reduces to the PQE problem on the subinstance I' of I whose Gaifman graph is G'.

We can use this idea to show our first intractability result, namely, the lower bound on d-SDNNF provenance representations. Indeed, we can show that, on some family $\mathcal{I}'$ of instances with planar degree-3 Gaifman graphs, the provenance of intricate queries must have treewidth proportional to that of the input instance, when expressed as a DNF. Up to a polynomial blowup, we can extract instances of $\mathcal{I}'$ in any sufficiently high treewidth instance, using [CC16]. We can then conclude using our result from [AMS18] which shows a lower bound on the size of d-SDNNF representations of high-treewidth DNFs of bounded arity and degree.

Our second intractability result is the hardness result, and it turns out to be more challenging to prove. We reduce from the #P-hard problem [XZZ07] of counting matchings on an input planar degree-3 graph H, and follow the reduction idea sketched above. We use the fact that the query Q_σ is intuitively satisfied on instances whose Gaifman graph contains two incident edges, i.e., is *not* a matching; so PQE for Q_σ amounts to weighted counting of matchings. Unfortunately, there is a significant technical obstacle: our oracle for PQE cannot be applied directly to count the matchings of the input graph H; we can only evaluate it on the subdivision G' extracted via the topological embedding from the high-treewidth instance of $\mathcal{I}$. Hence, we must argue that the weighted counting of matchings on the arbitrary subdivision G' (using our PQE oracle) can be used to recover the number of matchings on the original graph H.

To show this, we use the *interpolation method*. This is a general technique in reductions on counting problems [Gre00], where we subdivide the items that we want to count according to some value of a *parameter* — intuitively ensuring that all items having the same parameter value are easier to count together. Here, the items to count are the subgraphs of H. We then invoke an oracle for the target problem (here, PQE for Q_σ), on as many different inputs as there are parameter values — intuitively, these inputs are typically obtained from the original oracle input by varying the design of some gadget. Here, the original oracle input is I, and the variants are different choices of probabilities for I. We design these inputs to ensure that the vector of oracle results (indexed by the possible design variants) is related by a linear equation system to the vector of item counts that we want to recover (indexed by the possible parameter values). If we can show that the matrix of this system is invertible, then the vector of item counts can be obtained from the oracle answers simply by inverting the matrix.

In our reduction, the choice of parameter distinguishes the subgraphs of H by giving a type to the edges of H depending on how they are subdivided in G',

and counting the number of edges of each type. The gadget that we use in our oracle calls on I is simply a choice of probabilities on some edges in each path of G'. The crucial step to apply the interpolation method is to argue that we can choose suitable probabilities on the paths of G' that make shorter subdivisions intuitively "behave the same" as longer subdivisions, so that we can proceed as if all edges of H had been subdivided to paths of the same length. Further, as the requisite probabilities are irrational, we show that we can approximate them as rationals, and bound the error so as to ensure that the final result can be recovered by rounding.

5.4 Hardness for Unbounded Homomorphism-Closed Queries

We now move in this section to another direction of research on the PQE problem, started after my PhD and in collaboration with İsmail İlkan Ceylan. Like in the previous section, we restrict to arity-two signatures for technical reasons. Moving beyond CQs and UCQs, we study *recursive queries*, such as Datalog or regular path queries, on arbitrary TID instances. Our goal will be to establish that, for *any* such query, probabilistic query evaluation is #P-hard when allowing arbitrary TID instances as input. This should be contrasted with the results of the previous section, which apply to *some* queries, but show intractability for them on *any* restricted family of treewidth-constructible instances.

The problem of PQE for recursive queries had been previously studied in two different contexts. First, it has been investigated in the specific case of the *reliability* of probabilistic networks, in particular by Valiant [Val79]. Specifically, the *two-terminal reliability* problem [Val79] takes as input an undirected graph with two distinguished terminal vertices, where edges all have an independent probability of failure. It asks us to compute the probability in the resulting distribution that there is a path that connects the two terminals. This problem is #P-hard, which implies that PQE is intractable for the specific recursive query asserting that there exists a path connecting both terminals. Second, the research of Jung and Carsten [JL12] has shown intractability results for *ontology-mediated query answering* over probabilistic data. In this line of work, the query to evaluate is not expressed directly, but must be rewritten via (non-probabilistic) deduction rules expressed in a logical language from knowledge representation. Their work again implies that PQE is intractable for some specific recursive queries expressed in this way.

Of course, there are other recursive queries which enjoy tractable PQE, in particular the ones that are in fact logically equivalent to a tractable non-

recursive query, e.g., a safe UCQ [DS13]. To eliminate this phenomenon, we focus on understanding the status of PQE for what we call *unbounded queries*, namely, recursive queries that are *not* equivalent to a UCQ, and hence are not classified by [DS13]. We specifically study the unbounded queries that are *closed under homomorphisms*: this condition is satisfied by regular path queries, by Datalog queries (if they do not feature inequalities or negation), and by CQs and UCQs. Formally, a query Q is *closed under homomorphisms* if, whenever an instance I satisfies Q, and the instance I has a homomorphism to an instance I', then I' also satisfies Q. This implies in particular that the query is monotone, but is a stronger requirement, e.g., we cannot express inequalities.

To summarize, we study *unbounded homomorphism-closed queries*, i.e., queries which are homomorphism-closed and not equivalent to a UCQ. Equivalently, the query must have an infinite number of *minimal models*, i.e., models I of the query which are subset-minimal in the sense that no strict subinstance of I satisfies Q. We accordingly write UCQ^{∞} to denote the set of unbounded homomorphism-closed queries, because they can be equivalently expressed as an *infinite union* of CQs corresponding to their minimal models. Our result is the following:

Theorem 5.4.1 ([AC22], Theorem 4.3)**.** *Let Q be an unbounded query closed under homomorphisms on an arity-two signature. Then the probabilistic query evaluation problem for Q is #P-hard.*

Proof techniques. To show this result, we reduce from one of two #P-hard problems, depending on the query. The first problem is that of counting satisfying assignments of *positive partitioned 2-DNF formulas* [PB83] (#PP2DNF), i.e., counting non-independent sets in bipartite graphs. This is also the standard problem used to show the #P-hardness of PQE for non-hierarchical self-join-free CQs. The second problem is the *two-terminal network reliability* problem (U-ST-CON) mentioned earlier [Val79, PB83].

To design a reduction, the challenge is to understand the "behavior" of the query Q, i.e., which kind of instances satisfy Q, and how can Q become false when we modify these instances. One key way to modify instances is to perform a so-called *dissociation*: take one edge of the Gaifman graph, i.e., two distinct elements u and v that co-occur in some facts, and replace all such facts by two copies, each involving just one of u and v (see Figure 2). We say that an instance I satisfying Q has a *tight edge* if it has an edge on which the dissociation process yields an instance which no longer satisfies Q. Intuitively, a tight edge is a part of the instance whose connection to neighboring facts is in some sense necessary for the query to be true in that instance.

A key claim is then the following:

Figure 2: Dissociation of an edge (Figure 4 of [AC22])

Lemma 5.4.2 ([AC22], Theorem 6.6)**.** *On arity-two signatures, any query which is unbounded and closed under homomorphisms must have a model featuring a tight edge.*

Proof sketch. We use the fact that such queries have infinitely many minimal models; in particular, there are arbitrarily large minimal models. We take such a minimal model I and we repeatedly apply the dissociation process: we can see that this must eventually terminate. If one of the dissociations has made the query false, then we have found a tight edge. Otherwise, we have obtained a model of the query where no dissociation is possible, so its structure is very simple: it is intuitively a union of star-shaped patterns. We can show that such instances are homomorphically equivalent to a small (constant-sized) subset of facts, which we can then use to contradict the minimality of our large minimal model I. □

Note that this lemma is the only result using the unboundedness of the query: our hardness result in fact applies to any homomorphism-closed query that has a model with a tight edge, such as the non-hierarchical CQ $Q_0 : \exists x\, y\, R(x), S(x, y), T(y)$.

The tight edge can then be used to design gadgets that we can use for a reduction. The problem from which to reduce depends on whether the tight edge is *iterable*, intuitively, whether the query is still satisfied if we replace the edge by arbitrarily long "back-and-forth" paths. If the tight edge is not iterable, we can reduce from the #PP2DNF problem, like in [JL12]; if it is iterable, then we can reduce from U-ST-CON by using it to code the edges of a graph.

5.5 Hardness of Uniform Reliability

In this section, we review a third research direction on PQE, which studies the impact of another restriction on input instances: we restrict *which probabilities* are allowed on input TIDs. This is orthogonal to the structural restrictions on treewidth studied in Section 5.3, as these restrictions did not limit which probabilities were allowed on facts.

Of course, the PQE problem becomes trivially tractable when we allow only the probabilities 0 and 1 on input facts, as it then amounts to non-probabilistic query evaluation. However, almost all #P-hardness results on PQE mentioned so far are shown via reductions that crucially use facts with probability 1, as a convenient way to guarantee that the presence of the fact is certain. (The only exception is the proof of Theorem 5.3.3, which does not use probability 1, but uses many different probability values in the multiple oracle calls that it makes.) This leads to the question of whether PQE could become tractable if the probability 1 were disallowed, or more stringently if only the probability $1/2$ were allowed (in addition to the probability 0, which codes the absence of a fact).

The expected answer is that intractability should still hold under this restriction. However, the study of so-called *symmetric model counting* can cast some doubt on these expectations. In the context of PQE, symmetric model counting means that we require all *possible* facts over the domain over the input instance to have probability $1/2$, in particular we do not have facts with probability 0. The highly symmetrical nature of this problem was shown [BVdBGS15] to make PQE tractable for some previously intractable queries, e.g., the query $Q_0 : \exists x\, y\ R(x), S(x, y), T(y)$ mentioned earlier. Given this, it becomes more interesting to determine whether intractability really holds when all present facts are required to have probability $1/2$ (but when we still allow some facts to be missing).

We investigated this problem with Benny Kimelfeld [AK21], focusing on the *self-join free conjunctive queries* (SJFCQs) as a first step: indeed, these are the queries on which a dichotomy on PQE was first shown [DS07]. Our result covers in particular the case of input TIDs in which the probability of every (present) fact must be $1/2$. We call the corresponding problem *uniform reliability* (UR). As it is a special case of the PQE problem, we know that the UR problem is tractable for all queries enjoying tractable PQE. The challenging direction is to show the converse, i.e., that UR is already intractable whenever PQE is. We show the following dichotomy, which is the analogue of the PQE dichotomy [DS07] and uses the same notion of *hierarchical SJFCQs* (see [AK21, SORK11]):

Theorem 5.5.1 ([AK21], Theorem 3.1)**.** *Let Q be a SJFCQ. If Q is hierarchical, the PQE problem for Q, hence the UR problem for Q, is tractable. If Q is non-hierarchical, the UR problem for Q, hence the PQE problem for Q, is #P-hard.*

The tractability result follows from [DS07], so our contribution is to prove the second part of the statement. Our results more generally show a dichotomy on the so-called *weighted uniform reliability* problem, where we fix a constant probability p_R for each relation R of the signature, and impose that the (present)

facts of relation R must all have probability p_R. This extends an earlier dichotomy by Dalvi and Suciu [DS07] covering the setting where some relations are required to be deterministic (i.e., all present facts have probability 1).

Independently from our work, Suciu and Kenig have investigated the UR problem for the class of all UCQs [KS21] — they call it the *model counting problem*. Their results are far more technical than ours, but their conclusions are incomparable, as they focus on so-called type-I forbidden queries from [DS13]. In addition to UR, they also study *generalized model counting*, which only allows the probabilities 0, 1, and 1/2; they argue that this problem is more symmetric compared to UR which only allows 0 and 1/2. They show that the hardness of PQE for all unsafe UCQs of [DS13] already applies to this generalized model counting problem. In particular, this was already known for non-hierarchical SJFCQs — but this does not imply our claim that the UR problem for such queries is also intractable.

More recently, I have investigated in [Ama23] how to bridge the study of uniform reliability with the study of unbounded homomorphism-closed queries presented in the previous section. Specifically, I proved that Theorem 5.4.1 can be strengthened from PQE to UR:

Theorem 5.5.2 ([Ama23], Theorem 1.3)**.** *Let Q be an unbounded homomorphism-closed query on a graph signature. Then the UR problem for Q is #P-hard.*

Proof techniques. The result on uniform reliability for self-join-free CQs (Theorem 5.5.1) is shown by first limiting the study to the case of the intractable query $Q_0 : \exists x\, y\ R(x), S(x, y), T(y)$, using the standard methods of [DS07] to show hardness of non-hierarchical queries. On Q_0, the result is shown using the interpolation method reviewed in Section 5.3. Specifically, we do a variant of the usual reduction from the #PP2DNF problem, but add some number of parallel dangling edges to each vertex (intuitively controlling the number of R-facts and T-facts kept) and code S-facts by some number of parallel copies of a back-and-forth gadget.

As for the hardness result on unbounded homomorphism-closed queries (Theorem 5.5.2), it is a variant of the result for PQE (Theorem 5.4.1) and uses a similar proof structure, in particular it relies on Lemma 5.4.2. However, we now reduce from uniform variants of the #PP2DNF and U-ST-CON problems, using in particular the hardness of uniform reliability for Q_0 (Theorem 5.5.2). Further, we use a *saturation technique* of copying some non-unary facts a large number of times to make them behave as if their probability were 1, and we choose the tight edge more carefully by minimizing some notion of weight.

5.6 Perspectives

This chapter has reviewed the state of the art of the probabilistic query evaluation problem (PQE), and presented my contributions to its study, focusing on three areas: lower bounds on the size of provenance representations and hardness of the problem on unbounded-treewidth instances; #P-hardness of PQE on arbitrary unbounded homomorphism-closed queries; and #P-hardness of uniform reliability for the same class of queries and for non-hierarchical self-join-free CQs. All but the latter result apply only to arity-two signatures.

There are many open questions left about the PQE problem. One question is practical applicability: for this, the best exact approach currently available may be to efficiently compute provenance representations for queries (using a tool like ProvSQL [SJMR18]) and then computing the probability of query answers using efficient model counting software on the provenance representation (see [FHH21]). The question of tractable *approximations* of PQE also matters in practice, as one can expect real-world data to be annotated with probabilities that are themselves approximative (and perhaps not even normalized).

However, I believe that it is also worthwhile to pursue our study of PQE from a theoretical angle. These results may be of limited use to practitioners (in particular lower bounds), but they can hopefully appeal to research communities beyond database theory who study related topics: counting complexity [For97], holographic algorithms [Val08], as well as constraint satisfaction problems [Sch78, FV93, KZ17] and their counting variants [Bul13]. Let us accordingly review some further directions for theoretical research on the PQE problem.

Understanding the intractability of PQE. There are many further restrictions of PQE for which one can conjecture intractability, and hopefully prove it. I believe that some of these restrictions are especially interesting because they are arguably as natural as the original problem. This is in particular the case of uniform reliability (presented in Section 5.5): this simply asks, given a structure (or graph), *how many substructures* (or subgraphs) have a specific property (expressed as a Boolean query). In particular, studying this problem for the class of all queries closed under homomorphisms, one could conjecture:

Conjecture 5.6.1 ([Ama23], Conjecture 1.2)**.** *Given a query Q closed under homomorphisms, the uniform reliability problem is in PTIME if Q is equivalent to a safe UCQ, and #P-hard otherwise.*

The upper bound of this dichotomy is known, but showing the lower bound would require two new results: the hardness of uniform reliability for unsafe

UCQs (i.e., extending the result of Kenig and Suciu [KS21] to uniform reliability for all unsafe UCQs), and the hardness of uniform reliability for unbounded homomorphism-closed queries of arbitrary arity (generalizing [Ama23] to higher arity).

Going beyond queries closed under homomorphisms, it can of course be relevant to study the problem for other query classes, following the footsteps of initial works in this direction [FO16, OH08, OH09]. However, in this area, it is not clear which class of queries would be a realistic next step to achieve a dichotomy result.

Connecting PQE to provenance formalisms. One other important direction to better understand PQE is to characterize the power of the *intensional approach* reviewed in Section 5.3. Remember that, in this approach, we first compute provenance circuits for the query, and then argue that they fall in a tractable class of circuits (e.g., d-DNNF) so we can tractably compute the probability. Indeed, in the case of MSO query evaluation on trees and bounded-treewidth instances (i.e., the upper bounds sketched in Section 5.3), we could show the tractability of PQE using the intensional approach with d-SDNNF provenance circuits. Further, in the case of hierarchical SJFCQs [JS13], it is known that PQE can also be solved following the intensional approach using *read-once Boolean formulas* to represent provenance.

Interestingly, however, the algorithm for safe UCQs in [DS13] does *not* follow the intensional approach, and does not seem to be easily translatable to this setting. This is because the algorithm intuitively does not only do Boolean manipulations, but also uses the inclusion-exclusion formula, i.e., it does seemingly arbitrary arithmetic combinations of probabilities (including negative coefficients). This use of inclusion-exclusion is crucial to cover all safe queries, in particular it is necessary to avoid computing the probability of some subqueries that "cancel out" in the inclusion-exclusion formula. However, such computations have no obvious meaning in Boolean terms. The *intensional-extensional conjecture* thus asks whether the approach of [DS13] is strictly more powerful than the intensional approach.

Of course, the intensional-extensional conjecture depends on which class of provenance representations we allow. This question was studied by Jha and Suciu [JS13], which characterized which safe UCQs admitted tractable provenance representations as read-once formulas, and as OBDDs; they also achieved partial results for FBDDs and d-DNNFs. It was further shown by Beame et al. [BLRS17] that the class of so-called *Decomposable Logic Decision Diagrams* (DLDDs) could not apply to all safe UCQs, and by Bova and Szeider [BS17] that d-SDNNFs also did not suffice for safe UCQs. But the intensional-extensional conjecture still

stands for the more expressive class of (non-structured) d-DNNFs: that class was conjectured in [JS13] to be too weak, but the proposed counterexample was refuted by Monet [Mon20]. The conjecture also stands for the wider class of *d-Ds*, used by Monet [Mon20] to apply the intensional approach to a restricted class of safe UCQs: here d-Ds are simply d-DNNF circuits featuring arbitrary negations. Note that it is in fact open whether d-Ds strictly extend the power of the intensional approach compared to d-DNNFs, as we do not know whether one can tractably negate d-DNNFs [DM02].

In this light, I believe that the intensional-extensional conjecture for d-DNNFs and d-Ds is an open problem worthy of further study, in particular because of its connections to questions about knowledge compilation classes and about the combinatorics of the inclusion-exclusion formula.

The connection between the intensional and extensional approach is also a question in terms of lower bounds, e.g., when contrasting the two kinds of intractability results presented in Section 5.3 (lineage lower bounds, and computational hardness). Is there a unified approach to show both results? Note that, as far as we know, the two are incomparable: showing lower bounds on the size of provenance representations does not imply that PQE is intractable (because it may be solvable outside of the intensional approach); and conversely it could be the case that tractable provenance representations exist but that it is intractable to compute them and to solve PQE. A unified approach to hardness proofs and provenance lower bounds would also be useful, e.g., to understand if the hardness result of Section 5.4 also implies lower bounds on the size of provenance representations.

Proving joint tractability criteria on instances and queries. We have presented many intractability results known about PQE, and there are only a few settings where it is known to be tractable: safe UCQs on arbitrary instances [DS13], MSO queries on trees and bounded-treewidth data [ABS15, Ama16], and the case of symmetric model counting [BVdBGS15]. Neither of these results has broad practical applicability: safe UCQs are a very limited class, real-world data is not usually symmetric, and it also does not usually have bounded treewidth as was shown in an experimental study [MSJ19].

In light of this, it would be interesting to identify cases where we can guarantee tractability, even with an unsafe query and unbounded-treewidth data, by studying classes of queries and instances whose *interaction* is intuitively tractable. One first direction for this was the study in [DS07] of SJFCQs on (unbounded-treewidth) TIDs where some relations are required to be deterministic: this restriction can of course make more queries safe. Building on this approach, it would be interesting to investigate definitions of treewidth that

depend on a query, or on a query class, ensuring that PQE is tractable whenever this query-specific treewidth measure is bounded. Or, following the intensional approach: which joint restrictions on queries and databases can guarantee that we can efficiently compute tractable provenance representations? Such joint tractability criteria for provenance computation could then be useful, beyond PQE, for the various problems that can be addressed via provenance circuits, i.e., possibly also for enumeration and incremental maintenance.

Bibliography

This bibliography gives the DOI for research articles when it exists, and the publisher URL otherwise; as these link to the publisher version of articles, they may be unavailable without a subscription. By contrast, the article titles always link to an open-access version of the material (when one exists).

[ABC99] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999. doi:10.1145/303976.303983. (see page: 52)

[ABJM17] Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *ICALP*, 2017. doi:10.4230/LIPIcs.ICALP.2017.111. (see pages: 6, 8, 21, 22, 23, 24, 25, 26, and 33)

[ABM18] Antoine Amarilli, Pierre Bourhis, and Stefan Mengel. Enumeration on trees under relabelings. In *ICDT*, 2018. doi:10.4230/LIPIcs.ICDT.2018.5. (see pages: 6, 8, 40, and 41)

[ABMN19a] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT*, 2019. doi:10.4230/LIPIcs.ICDT.2019.22. (see pages: 6, 9, 21, and 30)

[ABMN19b] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In *PODS*, 2019. doi:10.1145/3294052.3319702. (see pages: 6, 8, 23, 33, 40, and 42)

[ABMN21] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. *TODS*, 46(1), 2021. doi:10.1145/3436487. (see pages: 6, 29, 33, and 34)

[ABR00] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In *SODA*, 2000. doi:10.5555/338219.338645. (see page: 37)

[ABS15] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, 2015. doi:10.1007/978-3-662-47666-6_5. (see pages: 6, 7, 54, 55, and 65)

[ABS16] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: Limits and extensions. In *PODS*, 2016. doi:10.1145/2902251.2902301. (see pages: 6, 7, 54, and 55)

[AC20] Antoine Amarilli and İsmail İlkan Ceylan. A dichotomy for homomorphism-closed queries on probabilistic graphs. In *ICDT*, 2020. doi:10.4230/LIPIcs.ICDT.2020.5. (see pages: 6, 53)

[AC22] Antoine Amarilli and İsmail İlkan Ceylan. The dichotomy of evaluating homomorphism-closed queries on probabilistic graphs. *LMCS*, 18, 2022. doi:10.46298/lmcs-18(1:2)2022. (see pages: 6, 59, and 60)

[AHR98] Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *FOCS*, 1998. doi:10.1109/SFCS.1998.743504. (see pages: 40, 48, and 50)

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995. ISBN: 978-0201537710. (see page: 38)

[AJMR22] Antoine Amarilli, Louis Jachiet, Martín Muñoz, and Cristian Riveros. Efficient enumeration for annotated grammars. In *PODS*, 2022. doi:10.1145/3517804.3526232. (see pages: 6, 9, 22, 31, and 33)

[AJP21] Antoine Amarilli, Louis Jachiet, and Charles Paperman. Dynamic membership for regular languages. In *ICALP*, 2021. doi:10.4230/LIPIcs.ICALP.2021.116. (see pages: 6, 9, 42, 44, 48, 49, and 50)

[AK21] Antoine Amarilli and Benny Kimelfeld. Uniform reliability of self-join-free conjunctive queries. In *ICDT*, 2021. doi:10.4230/LIPIcs.ICDT.2021.17. (see pages: 6, 10, and 61)

[AK22] Antoine Amarilli and Benny Kimelfeld. Uniform reliability of self-join-free conjunctive queries. *LMCS*, 2022. doi:10.46298/lmcs-18(4:3)2022. (see page: 6)

[AM22] Antoine Amarilli and Mikaël Monet. Weighted counting of matchings in unbounded-treewidth graph families. In *MFCS*, 2022. doi:10.4230/LIPIcs.MFCS.2022.9. (see pages: 6, 7, 55, and 56)

[AM23] Antoine Amarilli and Mikaël Monet. Enumerating regular languages in bounded delay. In *STACS*, 2023. To appear. Preprint: arXiv:2209.14878. (see page: 36)

[Ama16] Antoine Amarilli. *Leveraging the structure of uncertain data*. PhD thesis, Télécom ParisTech, 2016. (see pages: 26, 27, 54, 55, and 65)

[Ama23] Antoine Amarilli. Uniform reliability for unbounded homomorphism-closed graph queries. In *ICDT*, 2023. To appear. Preprint: arXiv:2209.11177. (see pages: 6, 10, 53, 62, 63, and 64)

[AMS18] Antoine Amarilli, Mikaël Monet, and Pierre Senellart. Connecting width and structure in knowledge compilation. In *ICDT*, 2018. doi:10.4230/LIPIcs.ICDT.2018.6. (see pages: 6, 7, 55, and 57)

[AP21] Antoine Amarilli and Charles Paperman. Locality and centrality: The variety ZG. Under review. Preprint: arXiv:2102.07724, 2021. (see pages: 6, 9, 46, and 47)

[Aui00] Karl Auinger. Semigroups with central idempotents. In *Algorithmic problems in groups and semigroups*. Springer, 2000. doi:10.1007/978-1-4612-1388-8_2. (see page: 45)

[AW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, 2014. doi:10.1109/FOCS.2014.53. (see page: 38)

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *SODA*, 2021. doi:10.1137/1.9781611976465.32. (see page: 29)

[Bag06] Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL*, 2006. doi:10.1007/11874683_11. (see pages: 8, 20, 21, and 26)

[BBL98] Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65(5), 1998. doi:10.1007/3-540-57155-8_248. (see page: 35)

[BDD+22] Pierre Bourhis, Laurence Duchien, Jérémie Dusart, Emmanuel Lonca, Pierre Marquis, and Clément Quinton. Pseudo polynomial-time top-k algorithms for d-DNNF circuits, 2022. Preprint: arXiv:2202.05938. (see page: 35)

[BDG07] Guillaume Bagan, Arnaud Durand, and Étienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *CSL*, 2007. doi:10.1007/978-3-540-74915-8_18. (see pages: 20, 38)

[BFG+17] Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, Tsvi Kopelowitz, and Pablo Montes. File maintenance: When in doubt, change the layout! In *SODA*, 2017. doi:10.1137/1.9781611974782.98. (see page: 49)

[BGJR21] Pierre Bourhis, Alejandro Grez, Louis Jachiet, and Cristian Riveros. Ranked enumeration of MSO logic on words. In *ICDT*, 2021. doi:10.4230/LIPIcs.ICDT.2021.20. (see page: 34)

[BGMP92] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *TKDE*, 4(5), 1992. doi:10.1109/69.166990. (see page: 52)

[BGQ+22] Marco Bucchi, Alejandro Grez, Andrés Quintana, Cristian Riveros, and Stijn Vansummeren. CORE: A complex event recognition engine. *PVLDB*, 2022. doi:10.14778/3538598.3538615. (see page: 34)

[BGS20] Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: A tutorial. *ACM SIGLOG News*, 7(1), 2020. doi:10.1145/3385634.3385636. (see page: 21)

[BH98] Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6), 1998. doi:10.1007/3-540-60084-1_80. (see pages: 8, 42)

[BKS17] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *PODS*, 2017. doi:10.1145/3034786.3034789. (see page: 38)

[BKS18a] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. *TODS*, 43(2), 2018. doi:10.1145/3232056. (see pages: 39, 48, and 50)

[BKS18b] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs under updates and in the presence of integrity constraints. In *ICDT*, 2018. doi:10.4230/LIPIcs.ICDT.2018.8. (see pages: 20, 38, 39, and 50)

[BLRS17] Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. Exact model counting of query expressions: Limitations of propositional methods. *TODS*, 42(1), 2017. doi:10.1145/2984632. (see page: 64)

[BM21] Christoph Berkholz and Maximilian Merz. Probabilistic databases under updates: Boolean query evaluation and ranked enumeration. In *PODS*, 2021. doi:10.1145/3452021.3458326. (see page: 53)

[Bod94] Hans L. Bodlaender. A tourist guide through treewidth. *Acta cybernetica*, 11(1-2), 1994. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3417. (see page: 11)

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewdith. *SIAM Journal on Computing*, 25(6), 1996. doi:10.1137/S0097539793251219. (see page: 26)

[BPV04] Andrey Balmin, Yannis Papakonstantinou, and Victor Vianu. Incremental validation of XML documents. *TODS*, 29(4), 2004. doi:10.1145/1042046.1042050. (see pages: 39, 40)

[Bri19] Karl Bringmann. Fine-grained complexity theory (tutorial). In *STACS*, 2019. doi:10.4230/LIPIcs.STACS.2019.4. (see page: 38)

[BS17] Simone Bova and Stefan Szeider. Circuit treewidth, sentential decision, and query compilation. In *PODS*, 2017. doi:10.1145/3034786.3034787. (see pages: 55, 64)

[BS19] Christoph Berkholz and Nicole Schweikardt. Constant delay enumeration with FPT-preprocessing for conjunctive queries of bounded submodular width. In *MFCS*, 2019. doi:10.4230/LIPIcs.MFCS.2019.58. (see page: 20)

[Bul13] Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *JACM*, 60(5), 2013. doi:10.1145/2528400. (see page: 63)

[BVdBGS15] Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu. Symmetric weighted first-order model counting. In *PODS*, 2015. doi:10.1145/2745754.2745760. (see pages: 61, 65)

[BVS22] Christoph Berkholz and Harry Vinall-Smeeth. A dichotomy for succinct representations of homomorphisms, 2022. Preprint: arXiv:2209.14662. (see page: 33)

[CC16] Chandra Chekuri and Julia Chuzhoy. Polynomial bounds for the grid-minor theorem. *JACM*, 63(5), 2016. doi:10.1145/2820609. (see pages: 7, 56, and 57)

[CGLS21] Nofar Carmeli, Martin Grohe, Peter Lindner, and Christoph Standke. Tuple-independent representations of infinite probabilistic databases. In *PODS*, 2021. doi:10.1145/3452021.3458315. (see page: 53)

[CIKW16] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *SIGMOD*, 2016. doi:10.1145/2882903.2912574. (see page: 52)

[CK20] Nofar Carmeli and Markus Kröll. Enumeration complexity of conjunctive queries with functional dependencies. *Theory of Computing Systems*, 64(5), 2020. doi:10.1007/s00224-019-09937-9. (see page: 21)

[CK21] Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. *TODS*, 46(2), 2021. doi:10.1145/3450263. (see pages: 20, 33, 34, and 39)

[CKS09] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009. doi:10.1145/1559795.1559831. (see page: 54)

[CLP23] Marco Console, Leonid Libkin, and Liat Peterfreund. Querying incomplete numerical data: Between certain and possible answers. In *PODS*, 2023. To appear. Preprint: arXiv:2210.15395. (see page: 53)

[CLR03] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, 2003. doi:10.1145/773153.773179. (see page: 52)

[Col91] Charles J. Colbourn. Combinatorial aspects of network reliability. *Annals of Operations Research*, 33(1), 1991. doi:10.1007/BF02061656. (see page: 52)

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1), 1990. doi:10.1016/0890-5401(90)90043-H. (see pages: 26, 54)

[Cou09] Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12), 2009. doi:10.1016/j.dam.2008.08.021. (see page: 20)

[CS19] Florent Capelli and Yann Strozecki. Incremental delay enumeration: Space and time. *Discrete Applied Mathematics*, 268, 2019. doi:10.1016/j.dam.2018.06.038. (see page: 21)

[CS21] Florent Capelli and Yann Strozecki. Enumerating models of DNF faster: Breaking the dependency on the formula size. *Discrete Applied Mathematics*, 303, 2021. doi:10.1016/j.dam.2020.02.014. (see page: 21)

[CS22] Nofar Carmeli and Luc Segoufin. Conjunctive queries with self-joins, towards a fine-grained complexity analysis, 2022. Preprint: arXiv:2206.04988. (see page: 33)

[CTG+21] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. In *PODS*, 2021. doi:10.1145/3452021.3458331. (see pages: 34, 35)

[CY12] Rada Chirkova and Jun Yang. *Materialized views*. Number 4 in Foundations and Trends in Databases. Now Publishers, Inc., 2012. ISBN: 978-1601986221. doi:10.1561/1900000020. (see page: 38)

[CZB+22] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Alessio Conte, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. *TODS*, 47(3), 2022. URL: https://arxiv.org/abs/1912.10704, doi:10.1145/3531055. (see page: 35)

[dA90] Assis de Azevedo. The join of the pseudovariety **J** with permutative pseudovarieties. In *Lattices, Semigroups, and Universal Algebra*. Springer, 1990. doi:10.1007/978-1-4899-2608-1_1. (see page: 45)

[Dar01] Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2), 2001. doi:10.3166/jancl.11.11-34. (see page: 24)

[DG07] Arnaud Durand and Étienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *TOCL*, 8(4), 2007. doi:10.1145/1276920.1276923. (see page: 20)

[DGNT21] Katrin M. Dannert, Erich Grädel, Matthias Naaf, and Val Tannen. Semiring provenance for fixed-point logic. In *CSL*, 2021. doi:10.4230/LIPIcs.CSL.2021.17. (see page: 17)

[DK21] Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. In *ICDT*, 2021. doi:10.4230/LIPIcs.ICDT.2021.5. (see page: 34)

[DKM+18] Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *JACM*, 65(5), 2018. doi:10.1145/3212685. (see page: 38)

[DM02] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *JAIR*, 17, 2002. doi:10.1613/jair.989. (see pages: 15, 22, and 65)

[DMRT14] Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for Datalog provenance. In *ICDT*, volume 3, 2014. doi:10.5441/002/icdt.2014.22. (see page: 17)

[dR95] Michel de Rougemont. The reliability of queries. In *PODS*, 1995. doi:doi/10.1145/212433.212479. (see page: 52)

[DRKT07] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *IJCAI*, volume 7, 2007. URL: http://ijcai.org/Proceedings/07/Papers/396.pdf. (see page: 51)

[DS07] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDBJ*, 16(4), 2007. URL: http://www.cs.washington.edu/homes/suciu/vldbj-probdb.pdf, doi:10.1007/s00778-006-0004-3. (see pages: 10, 52, 53, 61, 62, and 65)

[DS11] Arnaud Durand and Yann Strozecki. Enumeration complexity of logical query problems with second-order variables. In *CSL*, 2011. doi:10.4230/LIPIcs.CSL.2011.189. (see page: 34)

[DS13] Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *JACM*, 59(6), 2013. doi:10.1145/2395116.2395119. (see pages: 53, 55, 59, 62, 64, and 65)

[Fan15] Wenfei Fan. Data quality: From theory to practice. *ACM SIGMOD Record*, 44(3), 2015. doi:10.1145/2854006.2854008. (see page: 51)

[FG10] J. Flum and M. Grohe. *Parameterized complexity theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2010. ISBN: 978-3642067570. (see page: 15)

[FHH21] Johannes K. Fichte, Markus Hecher, and Florim Hamiti. The model counting competition 2020. *JEA*, 26, 2021. doi:10.1145/3459080. (see page: 63)

[FHM+95] Gudmund Skovbjerg Frandsen, Thore Husfeldt, Peter Bro Miltersen, Theis Rauhe, and Søren Skyum. Dynamic algorithms for the Dyck languages. In *Workshop on Algorithms and Data Structures*, 1995. doi:10.1007/3-540-60220-8_54. (see page: 48)

[FKP18] Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *PODS*, 2018. doi:10.1145/3196959.3196967. (see page: 28)

[FKRV15] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *JACM*, 62(2), 2015. doi:10.1145/2699442. (see pages: 8, 27, 28, and 29)

[FO16] Robert Fink and Dan Olteanu. Dichotomies for queries with negation in probabilistic databases. *TODS*, 41(1), 2016. doi:10.1145/2877203. (see pages: 53, 64)

[For97] Lance Fortnow. Counting complexity. *Complexity theory retrospective II*, 1997. URL: https://people.cs.uchicago.edu/~fortnow/papers/counting.pdf. (see page: 63)

[FRU+18] Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, 2018. doi:10.1145/3196959.3196987. (see pages: 8, 28, and 29)

[FS89] Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *STOC*, 1989. doi:10.1145/73007.73040. (see pages: 38, 43)

[FT98] Martin Farach and Mikkel Thorup. String matching in Lempel-Ziv compressed strings. *Algorithmica*, 20(4), 1998. doi:10.1007/PL00009202. (see page: 36)

[FV93] Tomás Feder and Moshe Y. Vardi. Monotone monadic SNP and constraint satisfaction. In *STOC*, 1993. doi:10.1145/167088.167245. (see page: 63)

[GFB94] Ming Gu, Martin Farach, and Richard Beigel. An efficient algorithm for dynamic text indexing. In *SODA*, 1994. URL: https://dl.acm.org/doi/10.5555/314464.314675. (see page: 37)

[GGH98] Erich Grädel, Yuri Gurevich, and Colin Hirsch. The complexity of query reliability. In *PODS*, 1998. doi:10.1145/275487.295124. (see page: 52)

[GHL+14] Robert Ganian, Petr Hliněný, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of MSO1 model-checking. *JCSS*, 1(80), 2014. doi:10.1016/j.jcss.2013.07.005. (see page: 56)

[GJ22] Étienne Grandjean and Louis Jachiet. Which arithmetic operations can be performed in constant time in the RAM model with addition?, 2022. Preprint: arXiv:2206.13851. (see page: 17)

[GJLS22] Moses Ganardi, Louis Jachiet, Markus Lohrey, and Thomas Schwentick. Low-latency sliding window algorithms for formal languages. In *FSTTCS*, 2022. To appear. Preprint: arXiv:2209.14835. (see page: 49)

[GKK+18] Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings. In *SODA*, 2018. doi:10.1137/1.9781611975031.99. (see page: 37)

[GKT07] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007. doi:10.1145/1265530.1265535. (see pages: 16, 17)

[GMS93] Ashish Gupta, Inderpal Singh Mumick, and Venkatramanan Siva Subrahmanian. Maintaining views incrementally. *ACM SIGMOD Record*, 22(2), 1993. doi:10.1145/170036.170066. (see page: 38)

[Gre00] Catherine Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *Computational Complexity*, 9(1), 2000. doi:10.1007/PL00001601. (see page: 57)

[GS18] Martin Grohe and Nicole Schweikardt. First-order query evaluation with cardinality conditions. In *PODS*, 2018. URL: https://arxiv.org/abs/1707.05945, doi:10.1145/3196959.3196970. (see page: 50)

[Hal17] Joseph Y. Halpern. *Reasoning about uncertainty*. MIT press, 2017. ISBN: 978-0262582599. (see page: 51)

[HD96] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *International journal of approximate reasoning*, 15(3), 1996. doi:10.1016/S0888-613X(96)00069-2. (see page: 54)

[Hen18] Monika Henzinger. The state of the art in dynamic graph algorithms. In *International Conference on Current Trends in Theory and Practice of Informatics*, 2018. doi:10.1007/978-3-319-73117-9_3. (see page: 37)

[HHH21] Kathrin Hanauer, Monika Henzinger, and Qi Cheng Hua. Fully dynamic four-vertex subgraph counting, 2021. Preprint: arXiv:2106.15524. (see page: 39)

[HHS22] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms–A quick reference guide. *JEA*, 2022. doi:10.1145/3555806. (see page: 37)

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, 2015. doi:10.1145/2746539.2746609. (see page: 38)

[HR98] Thore Husfeldt and Theis Rauhe. Hardness results for dynamic problems by extensions of Fredman and Saks' chronogram method. In *International Colloquium on Automata, Languages, and Programming*, 1998. doi:10.1007/BFb0055041. (see page: 48)

[ILJ84] Tomasz Imieliński and Witold Lipski Jr. Incomplete information in relational databases. *JACM*, 31(4), 1984. doi:10.1145/1634.1886. (see page: 52)

[IUV17] Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *SIGMOD*, 2017. doi:10.1145/3035918.3064027. (see page: 38)

[Jac] Louis Jachiet. On the complexity of a "list" datastructure in the RAM model. Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/46746 (version: 2020-05-01). (see page: 49)

[JL12] Jean Christoph Jung and Carsten Lutz. Ontology-based access to probabilistic data with OWL QL. In *ISWC*, 2012. doi:10.1007/978-3-642-35176-1_12. (see pages: 58, 60)

[JLSS14] Said Jabbour, Jerry Lonlac, Lakhdar Sais, and Yakoub Salhi. Extending modern SAT solvers for models enumeration. In *IRI*, 2014. doi:10.1109/IRI.2014.7051971. (see page: 22)

[JS05] HoonSang Jin and Fabio Somenzi. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In *DAC*, 2005. doi:10.1145/1065579.1065775. (see page: 22)

[JS13] Abhay Jha and Dan Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory of Computing Systems*, 52(3), 2013. doi:10.1007/s00224-012-9392-5. (see pages: 64, 65)

[KAK+14] Christoph Koch, Yanif Ahmad, Oliver Kennedy, Milos Nikolic, Andres Nötzli, Daniel Lupei, and Amir Shaikhha. DBToaster:

higher-order delta processing for dynamic, frequently fresh views. *VLDBJ*, 23(2), 2014. doi:10.1007/s00778-013-0348-4. (see page: 38)

[Kaz13] Wojciech Kazana. *Query evaluation with constant delay*. PhD thesis, École normale supérieure de Cachan-ENS Cachan, 2013. (see page: 50)

[KMMN22] Sarah Kleest-Meißner, Jonas Marasus, and Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras, 2022. doi:10.1145/3209108.3209144. (see pages: 8, 42)

[KNN+19] Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Counting triangles under updates in worst-case optimal time. In *ICDT*, 2019. doi:10.4230/LIPIcs.ICDT.2019.4. (see pages: 39, 50)

[KNOZ20] Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *PODS*, 2020. doi:10.1145/3375395.3387646. (see page: 39)

[KNOZ21] Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Machine learning over static and dynamic relational data. In *DEBS*, 2021. doi:10.1145/3465480.3467843. (see page: 38)

[KNOZ22] Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Conjunctive queries with output access patterns under updates, 2022. Preprint: arXiv:2206.09032. (see page: 39)

[KR95] Sanjiv Kapoor and Hariharan Ramesh. Algorithms for enumerating all spanning trees of undirected and weighted graphs. *SIAM Journal on Computing*, 24(2), 1995. doi:10.1137/S009753979225030X. (see page: 19)

[KS11] Wojciech Kazana and Luc Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science*, 7, 2011. doi:10.2168/LMCS-7(2:20)2011. (see page: 20)

[KS13] Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *TOCL*, 14(4), 2013. doi:10.1145/2528928. (see pages: 8, 20, 21, 26, and 50)

[KS21] Batya Kenig and Dan Suciu. A dichotomy for the generalized model counting problem for unions of conjunctive queries. In *PODS*, 2021. doi:10.1145/3452021.3458313. (see pages: 62, 64)

[KT10] Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of monadic second-order logic. In *LICS*, 2010. doi:10.1109/LICS.2010.39. (see page: 56)

[KW21] Paraschos Koutris and Jef Wijsen. Consistent query answering for primary keys in Datalog. *Theory of Computing Systems*, 65(1), 2021. doi:10.1007/s00224-020-09985-6. (see page: 52)

[KZ17] Andrei Krokhin and Stanislav Zivny. *The constraint satisfaction problem: Complexity and approximability*, volume 7. Schloss Dagstuhl, 2017. ISBN: 978-3959770033. (see page: 63)

[Lib04] Leonid Libkin. *Elements of finite model theory*. Springer, 2004. ISBN: 978-3540212027. doi:10.1007/978-3-662-07003-1. (see page: 13)

[LL09] Martin Lange and Hans Leiß. To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8(2009), 2009. URL: https://www.informaticadidactica.de/uploads/Artikel/LangeLeiss2009/LangeLeiss2009.pdf. (see page: 32)

[LM14] Katja Losemann and Wim Martens. MSO queries on trees: Enumerating answers under updates. In *CSL–LICS*, 2014. doi:10.1145/2603088.2603137. (see pages: 8, 40)

[Loh12] Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups-Complexity-Cryptology*, 4(2), 2012. doi:10.1515/gcc-2012-0016. (see page: 36)

[LP22] Carsten Lutz and Marcin Przybylko. Efficiently enumerating answers to ontology-mediated queries. In *PODS*, 2022. URL: https://arxiv.org/abs/2203.09288, doi:10.1145/3517804.3524166. (see page: 21)

[LS88] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series*

B (Methodological), 50(2), 1988. doi:10.1111/j.2517-6161.1988.tb01721.x. (see page: 54)

[Mar13] Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *JACM*, 60(6), 2013. doi:10.1145/2535926. (see page: 20)

[Mon20] Mikaël Monet. Solving a special case of the intensional vs extensional conjecture in probabilistic databases. In *PODS*, 2020. doi:10.1145/3375395.3387642. (see pages: 55, 65)

[MR22] Martín Muñoz and Cristian Riveros. Streaming enumeration on nested documents. In *ICDT*, 2022. doi:10.4230/LIPIcs.ICDT.2022.19. (see pages: 32, 33)

[MR23] Martín Muñoz and Cristian Riveros. Constant-delay enumeration for SLP-compressed documents. In *ICDT*, 2023. To appear. Preprint: arXiv:2209.12301. (see pages: 33, 36)

[MRV18] Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document spanners for extracting incomplete information: Expressiveness and complexity. In *PODS*, 2018. doi:10.1145/3196959.3196968. (see pages: 28, 29)

[MS19] Florin Manea and Markus L. Schmid. Matching patterns with variables. In *International Conference on Combinatorics on Words*. Springer, 2019. doi:10.1007/978-3-030-28796-2_1. (see page: 35)

[MSJ19] Silviu Maniu, Pierre Senellart, and Suraj Jog. An experimental study of the treewidth of real-world graph data. In *ICDT*, 2019. doi:10.4230/LIPIcs.ICDT.2019.12. (see page: 65)

[Mut05] Shanmugavelayutham Muthukrishnan. *Data streams: Algorithms and applications*, volume 1. Now Publishers, Inc., 2005. ISBN: 978-1933019147. (see page: 19)

[Nie18] Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras. In *LICS*, 2018. doi:10.1145/3209108.3209144. (see pages: 6, 8, 40, and 42)

[NS18] Matthias Niewerth and Luc Segoufin. Enumeration of MSO queries on strings with constant delay and logarithmic updates. In *PODS*, 2018. doi:10.1145/3196959.3196961. (see pages: 40, 41)

[OH08] Dan Olteanu and Jiewen Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, volume 5291, 2008. doi:10.1007/978-3-540-87993-0_26. (see pages: 53, 64)

[OH09] Dan Olteanu and Jiewen Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*, 2009. doi:10.1145/1559845.1559887. (see pages: 53, 64)

[OZ15] Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *TODS*, 40(1), 2015. doi:10.1145/2656335. (see pages: 22, 25)

[PB83] J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4), 1983. doi:10.1137/0212053. (see page: 59)

[PD06] Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4), 2006. doi:10.1137/S0097539705447256. (see page: 38)

[Pet19] Liat Peterfreund. *The complexity of relational queries over extractions from text*. PhD thesis, Technion, 2019. (see page: 28)

[Pet21] Liat Peterfreund. Grammars for document spanners. In *ICDT*, 2021. doi:10.4230/LIPIcs.ICDT.2021.7. (see pages: 9, 22, 30, and 31)

[PI97] Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *JCSS*, 55(2), 1997. doi:10.1006/jcss.1997.1520. (see pages: 38, 39)

[Pin86] Jean-Éric Pin. *Varieties of formal languages*. Foundations of Computer Science. Plenum Publishing Corp., New York, 1986. ISBN: 978-0946536122. doi:10.1007/978-1-4613-2215-3. (see page: 46)

[Pin19] Jean-Éric Pin. Mathematical foundations of automata theory. 2019. URL: https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf. (see page: 46)

[Pre17] Nicola Prezza. A framework of dynamic data structures for string processing. In *SEA*, 2017. doi:10.4230/LIPIcs.SEA.2017.11. (see page: 37)

[PT07a] Mihai Pătraşcu and Corina E. Tarniţă. On dynamic bit-probe complexity. *Theoretical Computer Science*, 380(1-2), 2007. doi:10.1016/j.tcs.2007.02.058. (see page: 43)

[PT07b] Mihai Pătraşcu and Mikkel Thorup. Randomization does not help searching predecessors. In *SODA*, volume 7, 2007. URL: http://people.csail.mit.edu/mip/papers/randpred/randpred.pdf. (see page: 49)

[RD00] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4), 2000. URL: http://sites.computer.org/debull/A00dec/rahm.ps. (see page: 52)

[Res18] IBM Research. SystemT, 2018. URL: https://researcher.watson.ibm.com/researcher/view_group.php?id=1264. (see page: 28)

[RS07] Christopher Re and Dan Suciu. Materialized views in probabilistic databases: For information exchange and query optimization. *PVLDB*, 2007. URL: https://vldb.org/conf/2007/papers/research/p51-re.pdf. (see page: 52)

[Rus03] Frank Ruskey. *Combinatorial generation*, volume 11. University of Victoria, Victoria, BC, Canada, 2003. Preliminary working draft. URL: https://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf. (see page: 19)

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *STOC*, 1978. doi:10.1145/800133.804350. (see page: 63)

[SdBBA19] Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In *NeurIPS*, 2019. URL: https://papers.nips.cc/paper/9318-smoothing-structured-decomposable-circuits. (see page: 24)

[Seg13] Luc Segoufin. Enumerating with constant delay the answers to a query. In *ICDT*, 2013. doi:10.1145/2448496.2448498. (see page: 20)

[Seg14] Luc Segoufin. A glimpse on constant delay enumeration. In *STACS*, 2014. doi:10.4230/LIPIcs.STACS.2014.13. (see page: 20)

[Seg15] Luc Segoufin. Constant delay enumeration for conjunctive queries. *ACM SIGMOD Record*, 44(1), 2015. doi:10.1145/2783888.2783894. (see page: 20)

[SFMS97] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *JACM*, 44(2), 1997. doi:10.1145/256303.256309. (see pages: 9, 39, 40, 43, 46, and 48)

[SJMR18] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. ProvSQL: Provenance and probability management in PostgreSQL. *PVLDB*, 11(12), 2018. doi:10.14778/3229863.3236253. (see pages: 34, 63)

[SM19] Yann Strozecki and Arnaud Mary. Efficient enumeration of solutions produced by closure operations. *DMTCS*, 21, 2019. doi:10.23638/DMTCS-21-3-22. (see pages: 30, 34)

[SORK11] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. *Synthesis lectures on data management*, 3(2), 2011. ISBN: 978-1608456802. (see pages: 52, 61)

[SS21a] Markus L. Schmid and Nicole Schweikardt. Spanner evaluation over SLP-compressed documents. In *PODS*, 2021. doi:10.1145/3452021.3458325. (see page: 36)

[SS21b] Markus L. Schmid and Nicole Schweikardt. A purely regular approach to non-regular core spanners. In *ICDT*, 2021. doi:10.4230/LIPIcs.ICDT.2021.4. (see page: 35)

[SSV22] Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. *JACM*, 69(3), 2022. doi:10.1145/3517035. (see pages: 20, 39, and 50)

[Str85] Howard Straubing. Finite semigroup varieties of the form V*D. *Journal of Pure and Applied Algebra*, 36, 1985. doi:10.1016/0022-4049(85)90062-3. (see page: 47)

[Str19] Yann Strozecki. Enumeration complexity. *Bulletin of EATCS*, 3(129), 2019. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/596/605. (see page: 21)

[SV17] Luc Segoufin and Alexandre Vigny. Constant delay enumeration for FO queries over databases with local bounded expansion. In *ICDT*, 2017. doi:10.4230/LIPIcs.ICDT.2017.20. (see pages: 20, 50)

[SVZ18] Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. *TODS*, 43(3), 2018. doi:10.1145/3241040. (see page: 49)

[SZ16] Thomas Schwentick and Thomas Zeume. Dynamic complexity: Recent updates. *ACM SIGLOG News*, 3(2), 2016. doi:10.1145/2948896.2948899. (see page: 38)

[TAG+20] Nikolaos Tziavelis, Deepak Ajwani, Wolfgang Gatterbauer, Mirek Riedewald, and Xiaofeng Yang. Optimal algorithms for ranked enumeration of answers to full conjunctive queries. *PVLDB*, 13(9), 2020. doi:10.14778/3397230.3397250. (see page: 34)

[TIAS77] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3), 1977. doi:10.1137/0206036. (see page: 19)

[Tor20] Szymon Toruńczyk. Aggregate queries on sparse databases. In *PODS*, 2020. doi:10.1145/3375395.3387660. (see page: 33)

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory*, 2(1), 1968. doi:10.1007/BF01691346. (see page: 14)

[Val75] Leslie G. Valiant. General context-free recognition in less than cubic time. *JCSS*, 10(2), 1975. doi:10.1016/S0022-0000(75)80046-8. (see page: 32)

[Val79] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3), 1979. doi:10.1137/0208032. (see pages: 19, 52, 58, and 59)

[Val08] Leslie G. Valiant. Holographic algorithms. *SIAM Journal on Computing*, 37(5), 2008. doi:10.1137/070682575. (see page: 63)

[vEBKZ76] Peter van Emde Boas, Robert Kaas, and Erik Zijlstra. Design and implementation of an efficient priority queue. *Mathematical systems theory*, 10(1), 1976. doi:10.1007/BF01683268. (see page: 46)

[Was16] Kunihiro Wasa. Enumeration of enumeration algorithms, 2016. (see page: 19)

[Weg00] Ingo Wegener. *Branching programs and binary decision diagrams*. SIAM, 2000. ISBN: 978-0898714586. (see page: 24)

[WG98] Tim A. Wagner and Susan L. Graham. Efficient and flexible incremental parsing. *TOPLAS*, 20(5), 1998. doi:10.1145/293677.293678. (see pages: 48, 50)

[XZZ07] Mingji Xia, Peng Zhang, and Wenbo Zhao. Computational complexity of counting problems on 3-regular planar graphs. *Theoretical Computer Science*, 384(1), 2007. doi:10.1016/j.tcs.2007.05.023. (see page: 57)

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, volume 81, 1981. URL: https://dl.acm.org/doi/10.5555/1286831.1286840. (see page: 38)