Counting, Knowledge Compilation and Applications

Stefan Mengel

Counting, Knowledge Compilation and Applications

Habilitation à diriger des recherches presented by

Stefan Mengel

at Université d'Artois
on December 8, 2021

Jury:
Pierre Marquis (Directeur), Universite d'Artois, France
Adnan Darwiche (Rapporteur), UCLA, US
Reinhard Pichler (Rapporteur), TU Wien, Austria
Pierre Senellart (Rapporteur), École normale supérieure, France
Nadia Creignou (Examinatrice), Aix Marseille Université, France
Hélène Fargier (Examinatrice), CNRS, France

To those who left

Contents

Acknowledgements

I would like to thank Adnan Darwiche, Reinhard Pichler and Pierre Senellart for accepting to review this thesis and their generous positive reviews. I would also like to thank them and Nadia Creignou, Hélène Fargier and Pierre Marquis for their participation in the jury and for putting up with the ever increasing administrative and pandemic related constraints involved in all of this. I thank Pasquale Mammone for his personal intervention to make my habilitation defense possible.

The research presented in this thesis was done during my postdoc at LIX and later as a member of CRIL. Both places have provided great environments for my research. I am thankful to all of my colleagues there for their help, their support and the general atmosphere.

Over the years, I have discussed research with many people, too many to name here. Sometimes this led to publishable results, sometimes not, but in any case I learned a lot from these interactions. I am thankful for all these encounters and discussions and for those who shared them with me.

Special thanks go to Arnaud Durand and Pierre Marquis. I consider both of them my mentors and friends and their influence on my scientific development cannot be overstated. I aim to one day pay forward their kindness and their advice.

None of this would have been possible without Hilke. I thank her for what she does and what she is.

Introduction

This habilitation thesis gives an overview and some details of most of my research roughly since my PhD in 2013. When I write "roughly" this is because there is no clear cut-off date: since the transition from my PhD studies to my postdoc was rather smooth, for some of my research it is not clear even to myself when the main work was done. Also, I followed some lines of work, in particular that with Hubie Chen on counting for (unions of) conjunctive queries, both during and after my PhD. So the choice of the cut-off for this thesis is rather arbitrary. In the end, I was conservative and decided to only present work here that was not even started before my postdoc at École Polytechnique. As a consequence, the trichotomy from [35] is presented neither in my PhD thesis nor here, and the same is true for my work on the arithmetic complexity of tensor contractions [36].

Since in the time period under consideration I have written quite a few papers and some of them are rather technical, I will not present all of my research in detail here. For most papers, I will only give an overview what they are about, give some context and leave the technicalities to the actual papers. Only for some contributions I also give technical details to give the reader a feeling for the type of work I have been doing and the tools and techniques I am using in my research. This keeps this thesis shorter and manageable for both the reader and the author. The interested reader is of course invited to find all the gory technical details of the results they would like to see in the papers, all of which should readily be found freely. If there is an arXiv version of papers, it should be seen as the authoritative version; in particular, the reader should not waste their time with versions offered by closed access publishers.

This thesis is written in two main parts: in Part I, I give an overview of my work by presenting the context and the main results of my publications. I also try to point out how they are connected and build on each other. Part II of this thesis gives the technical details of some of my work. The contributions I present there are chosen based on different criteria: I chose some of my work that I found the most "important"—whatever this means in the context of purely theoretical work—and at the same time I wanted to present a certain diversity in what I am doing. That is why I chose papers from different areas, using different techniques to yield different types of results. The hope is that the reader will thus find a certain breadth in my work.

Most of the research presented here deals with counting complexity and knowledge compilation, two intimately related areas whose different facets appear in diverse areas of computer science. Since these two areas cover the bulk of my work, I have decided to restrict myself to them to make this thesis at least somewhat coherent.

Counting complexity is certainly a more well-known area than knowledge compilation: the main idea is that, in contrast to complexity classes of decision problems like NP, one does not only want to decide if there *is* a solution to a given problem instance but

wants to count *how many* such solutions there are. This was pioneered in classical work by Valiant [Val79a], who defined the complexity class #P generalizing the definition of the class NP. Essentially, instead of asking if there *exists* a certificate for an instance accepted by a polynomial time verifier, one asks how many such certificates there are. This question is often very natural. For example, standard practical database query languages have a language construct that allows counting answers to queries—which are the certificates of a query being true on a given database. But apart from being a natural question in itself, there are many applications of counting in areas that deal with probabilities, such as probabilistic reasoning in artificial intelligence or probabilistic databases. In these settings, the certificates are states of a system that satisfy a desired property. Since most of the time counting all possible states is easy, knowing the number of states satisfying the property directly yields the probability of the property being satisfied. [1]

I have contributed to counting complexity in two areas: propositional model counting, i.e., the question of counting models of a given propositional formula, and counting the answers to database queries, given a query and a database instance. Both of these problems are well known to be intractable in general, even though in practice they are sometimes solvable efficiently. For example there has been a certain progress practical in propositional model counting, see e.g. the results of the recent model counting competition [FHH20]. My work focused on ways of refining the general hardness results by getting an understanding of tractable fragments which in some cases even leads to complete dichotomy theorems. The general idea—which is pervasive in much of my work also beyond counting complexity—is that there is often a natural way of assigning graphs to problem instances. Now when those graphs are "simple", their structure can often be used to solve the problem at hand, in this case to count certificates for an instance, more efficiently than for general instances. Sometimes one can even completely determine for which kinds of graphs this approach can work. I will discuss my work on counting in Chapter 1.

The other main area of my research is knowledge compilation, a field that was originally introduced in artificial intelligence as a preprocessing regime to deal with hard reasoning problems as they appear in abundance in that area. The idea was that one could in certain settings use costly preprocessing on an offline knowledge base that could then be queried more efficiently in a second, online phase. We will not go into more detail here and refer the interested reader to the survey [Mar15] since almost none of the original motivation for knowledge compilation will play any role in this thesis. This is because knowledge compilation quickly after its inception turned towards the systematic study of data structures that knowledge could be translated, or as it is called *compiled*, into. This study of data structures, or *languages* as they are often called, is what we will mostly consider. In particular, we will almost exclusively deal with *circuits in decomposable negation normal form (DNNF)* which since their introduction by Adnan Darwiche now

[1] The attentive reader might remark that this assumes that the possible states are uniformly distributed. We will not go into this in detail here but only remark that other settings can often be reduced to this. Moreover, it is known that many counting algorithms also work for *weighted* counting which allows natively expressing probabilities even for non-uniform distributions.

roughly 20 years ago [Dar01a] have been one of the main objects of study in knowledge compilation.

The substantial amount of research that was done on DNNF over the years is mostly explained by the fact that they form a superclass of many other representation languages such as OBDD, FBDD, DNF and SDD. Thus they are a useful abstraction of all those. Moreover, there are systematic ways of getting useful subclasses of DNNF by requiring properties such as "determinism" or "structure" which we will explain later. Somewhat unexpectedly, (subclasses of) DNNF have also found a life beyond the original knowledge compilation setting that they were originally thought for. Essentially, they tend to come up often when knowledge/constraints/data/... have to be encoded in a concise but usable way. For example, they have been used for uncertain data in databases, to encode and decompose constraints in constraint programming, or to represent Bayesian classifiers; see e.g. [Mon20, JBKW08, KS19, CD03] and the references therein for an entry point into these directions of research. It is also understood now that DNNF can be seen as the traces of different algorithms, solving problems such as propositional model counting [HD07, 33] or runs of automata on words or trees [25, FRU$^+$18, 20]. There is also a whole range of practical compilers creating DNNF from functions represented in other formats, mostly CNF formulas [Dar04, Dar11, MMBH12, CD13, OD15, LM17a].

Here we will mostly ignore all practical considerations of DNNF and their construction and instead focus exclusively on theoretical aspects. We will first survey my contributions to what I consider core knowledge compilation questions in Chapter 2: there are some contributions to the so-called knowledge compilation map. Moreover, there is a compilation algorithm that allows using underlying graph structure of CNF-formulas, as discussed above for counting. Then I will present an overview of techniques to show lower bounds for DNNF that we introduced in [27] and which have since then found wide application by myself and others. These lower bound techniques will play a central role in the rest of this thesis.

After the chapter on core knowledge compilation questions, I will present in Chapter 3 applications of knowledge compilation, in fact mostly (subclasses of) DNNF, to other areas. We will see uses of DNNF for solving quantified Boolean formulas, for the proof complexity of propositional formulas and quantified Boolean formulas, and for bounds for width measures of CNF-encodings. Finally, there will be a section on so-called constant delay enumeration algorithms in database theory for which we used variants of DNNF as a crucial data structure to conceive efficient algorithms. DNNF were originally not invented for any of the above applications and some of them might be surprising. However, it turns out that in all these settings DNNF provide a good framework to reason about succinctly represented knowledge that can still be used efficiently.

References. Throughout this thesis, to make it easy to differentiate between my work and that of others, I am using two different citation styles: on the one hand, all works which I have contributed to as an author, are cited numerically such as [27]. These references can be found in my CV in Appendix A. On the other hand, all works which I have *not* contributed to are referenced alphabetically, such as [DM02] and can be found in the reference section at the end of this thesis.

The authors list of all papers I have contributed to is ordered alphabetically, as it is common in all areas I have worked in. In particular, from the position of the authors the reader cannot infer any information on the contribution of any author. It is safe to assume that all authors have contributed roughly equally. While this might not always be correct for each individual paper, on average it is probably close to being true.

Part I.

Overview

This part of this thesis is a survey of most of my work since my PhD. I try to keep this relatively concise. In particular, I leave out all technical details and focus instead of the context and the contribution of the individual works. Technical details of some contributions can be found in Part II. Here, I will first discuss my results on counting complexity in Chapter 1 and then turn to knowledge compilation in Chapter 2 and Chapter 3.

1. Counting

Counting complexity was a central part of my PhD thesis and it is still an area that I have a particular interest in. I have worked in two different areas which I will treat independently in the following: propositional model counting and counting for (unions of) conjunctive queries in database theory.

1.1. Propositional Model Counting

The propositional model counting problem #SAT is the problem of counting the models of a given propositional formula, generally assumed to be in CNF. #SAT is the generic complete problem for #P and as such plays a role similar to that of SAT for NP. However, in general #SAT is far harder. For example, thanks to the classical work of Toda [Tod91], it is known that for every problem in the polynomial hierarchy there is a polynomial time algorithm with one oracle call to a #SAT-oracle. That is, if we allow an algorithm to get the answer of a single #SAT-instance "for free", then the rest of the computation can be performed in polynomial time. Since the polynomial hierarchy contains many important problems presumably far harder than NP, this shows that solving #SAT must in fact be very hard. It is also known that #SAT is hard to approximate: unless $\mathsf{P} = \mathsf{NP}$, there is no polynomial-time approximation algorithm for #SAT with a guaranteed approximation factor of $2^{n^{1-\varepsilon}}$ where $\varepsilon > 0$ and n is the number of variables of the input [Rot96]. This remains even true if the input is restricted to classes like 2-CNF or Horn-CNF for which the decision problem is easy.

As a reaction to these general hardness results for #SAT, there has been intensive research on the question of finding restrictions which make the problem tractable. One of these directions is restricting the individual conjuncts of the input. This leads to the problems #CSP[Γ] for every set of Boolean relations Γ which is, given a conjunction F of atomic formulas in relations from Γ, to count the satisfying assignments of F. As already said above, for some sets of relations Γ, e.g. those defined by clauses in two variables, #CSP[Γ] is hard. On the other hand, there are some sets of relations, say all unary relations, for which #CSP[Γ] is easy. Creignou and Hermann [CH96] showed that in the Boolean case there is in fact a dichotomy theorem that determines which sets Γ lead to hard #CSP[Γ] and for which Γ the problem becomes easy. This was generalized to non-Boolean domains by Bulatov [Bul13] and Dyer and Richerby [DR13].

A different approach to finding tractable fragments for #SAT is instead of restricting the individual constraints of instances, so in a sense the "building blocks", to restrict the structure of the instances, so how the building blocks can be put together. The most common approach is assigning a graph to every CNF-formula and then developing

algorithms that rely on the structure of these graphs. For example, it is known that if the associated graphs are of bounded treewidth, then there are efficient model counting algorithms, see e.g. [SS10a], but there are many more general width measures for CNF which lead to tractable #SAT. A hierarchy of such results is shown in Figure 1.1.

I wrote two papers that are related to this approach and which are motivated by notions from database theory. Popular width measures for graphs like treewidth are often generalizations of being a tree—or more precisely, being acyclic because connectivity is often secondary—and measure how close to being acyclic a graph is. However, instead of a graph, one can also assign a hypergraph to a CNF-formula in a natural way: the vertices are the variables of the formula and for every clause there is an edge containing the variables of the clause. When one now wants to generalize algorithms for acyclic graphs to hypergraphs, one is faced with the problem that it is not clear what *acyclicity* is even supposed to be for hypergraphs. In fact, there are several notions of acyclicity that have been studied for a long time in database theory and that are useful in different settings [Fag83, Bra16]. So which of these acyclicity notions are useful to give tractable fragments for #SAT?

On the one hand, it was known from [SS10a] that α-acyclicity, the most general hypergraph acyclicity notion from the literature, is not restrictive enough to make #SAT tractable. On the other hand, from [SS13, GP04] it follows that #SAT for so-called γ-acyclic CNF-formulas is tractable. This left the open question for the intermediate case of so-called β-acyclicity. This case was not only particularly interesting because it was the main acyclicity notion not understood in the setting. Beyond this, it was also the only known structural restriction where there was an efficient algorithm for SAT that did not directly generalize to give a counting algorithm. This was because because the known algorithm from [OPS13] was unlike all other algorithms in the area not based on dynamic programming but on resolution. Since there are classes of CNF-formulas which allow efficient decision by resolution but which are hard for counting, e.g. 2-CNF, this hinted towards the fact that something interesting should happen for β-acyclic #SAT: either it would the only known structurally restricted class of CNF-formulas for which counting was hard but decision easy or there might be a new counting algorithm generalizing resolution proofs.

In a first paper in this direction with Arnaud Durand and Florent Capelli [36], we considered hypergraphs with *disjoint branches decompositions* which are a class of hypergraphs that lies strictly between γ-acyclic and β-acyclic hypergraphs. We showed that the propositional model counting problem #SAT for CNF-formulas with hypergraphs that allow a disjoint branches decomposition can be solved in polynomial time. We also showed that this class of hypergraphs which was introduced by Duris in [Dur12] is incomparable to hypergraphs of bounded incidence cliquewidth which were the biggest class of hypergraphs for which #SAT was known to be solvable in polynomial time at the time. Furthermore, we presented a polynomial time algorithm that computes a disjoint branches decomposition of a given hypergraph if it exists and rejects otherwise. Finally, we showed that some slight extensions of the class of hypergraphs with disjoint branches decompositions lead to intractable #SAT, leaving open how to generalize the counting result of this paper.

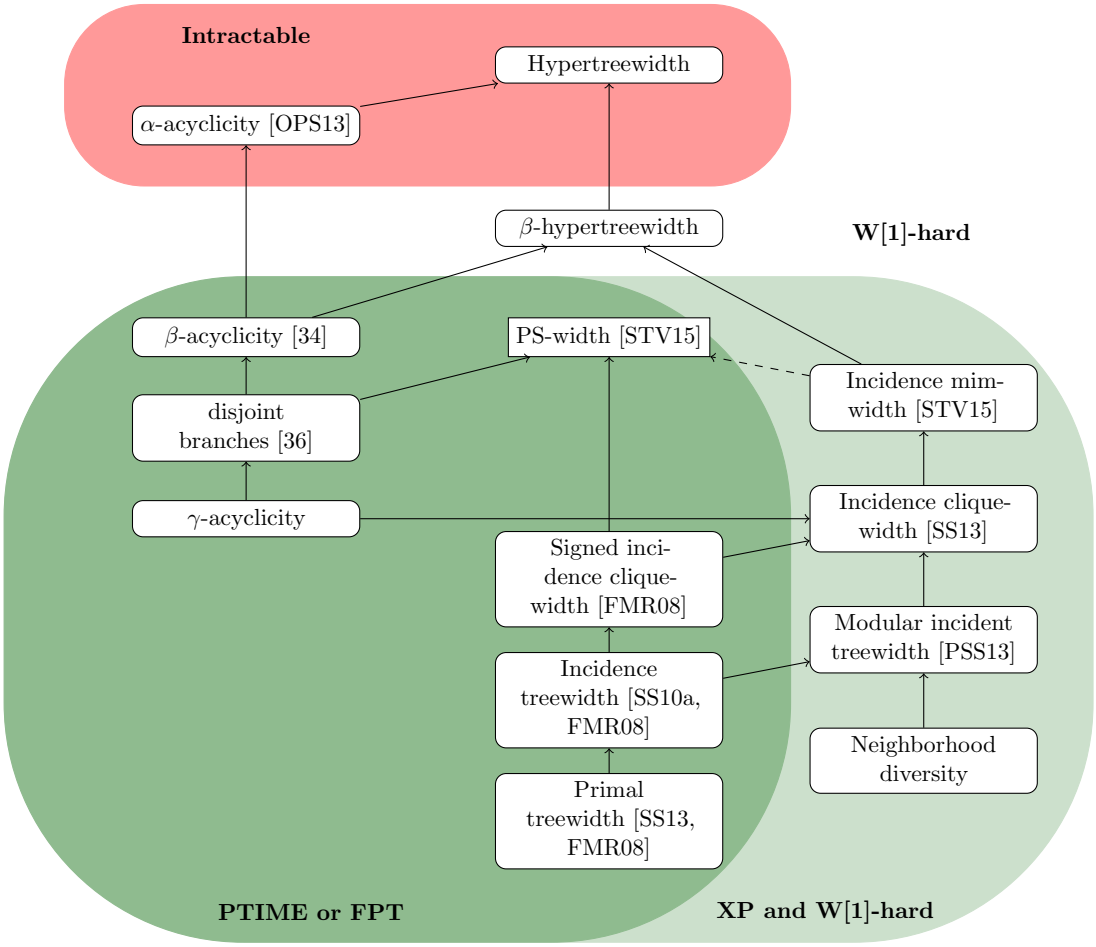


Figure 1.1.: Several restrictions of graphs, resp. hypergraphs, assigned to CNF-formulas that make the problem #SAT tractable. On the left, several different hypergraph acyclicity notions are given while the rest of the restrictions are restrictions of graphs associated to formulas. Arrows go from more restrictive to more general notions. The restrictions in the red bubble leave #SAT #P-hard. Parameters on the dark-green background allow FPT-algorithms, more specifically algorithms with runtime $2^{O(k)}\text{poly}(n)$ where k is the parameter and n the instance size. The parameters on the light-green background allow runtimes with runtimes of roughly $n^{O(k)}$ which cannot be significantly improved under standard complexity assumptions.

In [34], a joint paper with Johann Brault-Baron and Florent Capelli, we finally solved the case of β-acyclic formulas by showing that #SAT on them can be solved in polynomial time. In contrast to previous algorithms for other structurally restricted classes of formulas, our algorithm does not proceed by dynamic programming. Instead, it works along an elimination order, solving a weighted version of constraint satisfaction by a resolution-like process. We also give evidence that this deviation from more standard algorithms is no coincidence by showing that β-acyclic formulas are outside of the framework that was (previously to our paper) proposed by Saether et al. [STV15] and which subsumes all other structural tractability results for #SAT known so far.

Some of the ideas used in [34] have been further developed in [6], a paper studying some questions purely in structural graph theory: to show that the counting algorithm of [34] lies outside of the framework of [STV15], one technical ingredient is showing that there are so-called chordal bipartite graphs have unbounded mim-width. Here mim-width is a relatively recent graph width measure that was introduced in [Vat12] and that has seen applications mostly in graph algorithms. In [STV15], it was used in the context of propositional satisfiability, generalizing many previous results for more restrictive width measures. In [6], I showed how to lift the lower bounds on mim-width from [34] to lower bounds for the mim-width of strongly chordal split graphs, co-comparability graphs and circle graphs. This improved and refined lower bounds that were known before, some of them only conditionally, and answered several questions of Vatshelle [Vat12]. In the case of strongly chordal graphs not even a conditional lower bound was known before. All of the bounds given are optimal up to constant factors.

1.2. Unions of Conjunctive Queries

The computational problem of evaluating a formula (of some logic) on a finite relational structure is central in database theory and logic. In the context of database theory, this problem is often referred to as *query evaluation*, as it models the posing of a query to a database: the formula is the query, and the structure represents the database. The results of such an evaluation are called *answers*; logically, these are the satisfying assignments of the formula on the structure. The particular case of this problem where the formula is a sentence, i.e., when it has no free variables, is often referred to as *model checking*, and even in just the case of first-order sentences, can capture a variety of well-known decision problems from all throughout computer science [FG06].

I wrote several papers on the counting version of this problem, namely, given a formula and a structure, output the *number* of answers. This problem of *counting query answers* generalizes model checking, which can be viewed as the particular case thereof where one is given a sentence and a structure, and wants to decide if the number of answers is 1 or 0, corresponding to whether or not the empty assignment is satisfying. In addition to the counting problem's basic and fundamental interest, all practical query languages supported by database management systems have a counting operator.

A typical situation in the database setting is the evaluation of a relatively short formula on a relatively large structure. Consequently, it has been argued that, in measuring

the time complexity of query evaluation tasks, one could reasonably allow a slow (non-polynomial-time) preprocessing of the formula, so long as the desired evaluation can be performed in polynomial time following the preprocessing [PY99, FG06]. Relaxing polynomial-time computation to allow arbitrary preprocessing of a *parameter* of a problem instance yields, in essence, the notion of *fixed-parameter tractability*. This notion of tractability is at the core of *parameterized complexity theory*, which provides a taxonomy for classifying problems where each instance has an associated parameter. In my work, I made use of this paradigm using the formula as the parameter.

During my time as a PhD-student, I had worked extensively on the counting complexity of *conjunctive queries*, a foundational and practically very important class of conjunctive queries, see [Men13] for an overview. After my PhD, I extended this line of work in two joint papers with Hubie Chen [28, 26].

Existential positive queries are the first-order formulas built from the two binary connectives $(\wedge, \vee)$ and existential quantification. They include and are semantically equivalent to the so-called *unions of conjunctive queries*, also known as *select-project-join-union queries*; these have been argued to be the most common database queries [AHV95]. Indeed, each union of conjunctive queries can be viewed as an existential positive query having a particular form, namely, a disjunction of primitive positive formulas; recall that a *primitive positive* query is an existential positive query that does not use disjunction.

In [28, 26], we study the complexity of counting query answers on existential positive queries. An established way to understand which types of queries are computationally well-behaved and exhibit desirable, tractable behavior is to consider this problem relative to a set of queries, and to attempt to understand on which sets this problem is tractable. Precisely, each set Φ of existential positive queries yields a restricted version of the general problem, namely: count the number of answers of a given formula $\phi \in \Phi$ on a given finite structure $\mathbf{B}$. There is thus a family of problems, one problem for each such set Φ. Our study focuses on formula sets that have *bounded arity* (by which is meant that there is a constant that upper bounds the arity of all relation symbols used in formulas); we will assume this property of all formula sets throughout this section.[1]

In [28], we prove a trichotomy theorem on the parameterized complexity of the discussed family of problems, which describes the complexity of every such problem. In particular, our trichotomy theorem shows that—in a sense made precise—each such problem is fixed-parameter tractable, equivalent to the *clique* problem, or as hard as the *counting clique* problem (which generalizes the clique problem). Note that the hypothesis that the clique problem is not fixed-parameter tractable is an established one in parameterized complexity;[2] under this hypothesis, our trichotomy theorem yields a precise description of the problems that are fixed-parameter tractable. Our trichotomy theorem is in fact derived by invoking two theorems:

- A new theorem showing that, for each set of existential positive queries, there exists a set of primitive positive queries such that the two sets exhibit the same

[1] Note that in the case of unbounded arity, complexity may depend on the choice of representation of relations [CG10].

[2] It can be phrased in terms of complexity classes as FPT $\neq$ W[1].

complexity behavior. This new theorem, which we call the *equivalence theorem*, can be conceived of as the primary technical contribution of this article.

- A previously presented trichotomy on primitive positive queries from our previous work [35].

The statement of our new trichotomy theorem generalizes, unifies, and strengthens a number of existing parameterized complexity classification results in the literature [Gro07, GSS01, Che14a, DJ04a, 35, 38].

The techniques used to prove our equivalence theorem are algebraic and combinatorial, and are quite different in nature from and contrast with those used to prove the previous classifications, which were more graph-theoretic and logical in flavor. Indeed, while the graph-theoretic measure of *treewidth* played a key role in the statement and proof of the previous trichotomy as well as of the previous dichotomies on primitive positive queries, it is not at all needed to prove our equivalence theorem. To establish the equivalence theorem, we make a key application of the inclusion-exclusion counting principle to translate an existential positive formula to a finite set of primitive positive formulas, which, in the setup considered by the article, is crucial to handling and understanding disjunction. We remark that a very similar technique was independently developed by Curticapean, Dell and Marx [CDM17] in the setting of subgraph counting.

In [26], we give a syntactic version of the trichotomy in [28]. The motivation is that with a first-order formula ϕ in hand, if one is interested in counting the number of answers to ϕ on given structures, it is natural to inquire if there is a language or logic in which one can directly express the mapping that provides, for each structure, the number of answers to ϕ. Such a logic could serve as a target language into which first-order formulas of interest (in the mentioned sense) could be compiled, and then optimized, rewritten, and evaluated. The paper [26] presents and studies such a logic, $\sharp$-*logic*, in which the evaluation of a sentence on a structure yields an integer value. From the database-theoretic viewpoint, our presentation of $\sharp$-logic amounts to the introduction of a query language designed particularly for counting answers. We show that $\sharp$-logic enjoys and balances the following properties:

- **Expressiveness.** In a sense made precise, $\sharp$-logic allows for the expression of known efficient algorithms for tractable cases of the counting query answers problem. Moreover, this expression is (in our view) direct and clean, and illustrates that $\sharp$-logic captures precisely the key computational primitives required by these algorithms.

- **Optimizability.** Minimizing a crucial measure known as *width* can be performed computably in an expressive fragment of $\sharp$-logic; this amounts to the fragment supporting an optimal form of query optimization, relative to this quantity.

We show that width measure that we associate to $\sharp$-logic characterizes exactly the frontier of tractability for existential positive formulas: on the one hand, if a class of existential positive formulas Φ can be translated into bounded width $\sharp$-formulas, then they can be evaluated efficiently. The other way round, let Φ be any class of existential

positive queries having bounded arity. If counting answers to formulas in Φ is tractable, then the formulas in Φ can be translated into formulas in $\sharp$-logic that evaluate to the correct output on all finite structures. That is, having bounded width $\sharp$-formulas is the *exclusive* explanation for the tractability of counting answers to formulas in Φ in this existential positive setting. On a conceptual level, we view this result as strong evidence that, for the problem of counting query answers, $\sharp$-logic is a useful, expressive model of computation in which relevant, efficient algorithms can be presented. This result is obtained as an immediate consequence of two theorems:

- We show that when the counting problem for Φ is tractable, then there exists a bounded width class Ψ of $\sharp$-sentences such that each $\phi \in \Phi$ has a representation in Ψ.
- We prove that there is a *minimization algorithm* that, given an existential positive formula, computes a representation of minimum width.

The latter theorem, which we view as a key contribution in and of itself, can be read as demonstrating that $\sharp$-logic is well-characterized and well-understood as a model of computation: conceiving of a $\sharp$-sentence representation of an existential positive formula as a computational procedure for counting query answers, this theorem provides a minimization algorithm that always outputs an *optimal* procedure for a given existential positive formula, where optimality here is measured using width.

2. Knowledge Compilation

My contribution to the field of knowledge compilation has been rather wide. I have contributed to extensions of the knowledge compilation map, proposed compilation algorithms, and shown lower bounds. Besides this work inside knowledge compilation, one focus of my work has been applying knowledge compilation in contexts that it was not originally thought for. To give all this a little structure, I will present all contributions that are on core questions in knowledge compilation in this chapter. In Chapter 3, I will then present the applications outside of this core area.

2.1. Contributions to the Knowledge Compilation Map

One of the classical objects of study of knowledge compilation is classifying representation languages along the criteria of the so-called knowledge compilation map. This framework, introduced in the ground-breaking work of Darwiche and Marquis [DM02] gives a list of standard properties which should be analyzed for languages used in the area of knowledge compilation along different axes: *succinctness*, *queries* and *transformations*. Here succinctness measures the relative size of knowledge representations in different languages. When considering queries, one asks which standard problems on encoded knowledge—e.g. consistency and validity checks, counting but also enumeration—can be performed efficiently when the knowledge is encoded in the representation language considered. Here it is important to note that the runtime is measured relative to the size of the encoding. As a consequence, the same query on the same knowledge must be solved in less time on succinct representations than on verbose ones to be considered efficiently solvable. This puts succinct datastructures at a disadvantage for efficient query evaluation and as one consequence it has been observed that there is generally a trade-off between succinctness and queries: succinct representation languages allow less queries to be performed on them efficiently while more verbose data structures allow more efficient queries. The third axis of the knowledge compilation map finally are transformations. Here the question is in which way knowledge in a given representation can be transformed efficiently, for example by conditioning or forgetting variables, conjoining or negating knowledge. These standard transformations are often useful when using knowledge compilation as a building block in some larger problem but also in compilation itself when conjoining is used for bottom-up compilation as this is often the case for representations such as OBDD [Bry86, Som09] and SDD [Dar11, CD13].

The idea of the knowledge compilation map has had a huge influence and the approach of [DM02] is widely applied in knowledge compilation, see e.g. [PD08, FM14, FMN13] for a very small sample. I contributed some work in this direction which I will discuss in

this section.

In joint work with my student Romain Wallon and his other two PhD supervisors Daniel Le Berre and Pierre Marquis [23], we study pseudo-Boolean constraints (PBC), i.e. integer inequalities over $\{0, 1\}$, and their special case cardinality constraints (CARD) along the criteria of the knowledge compilation map. To this end, we compare the succinctness of PBC and CARD to that of many standard propositional languages studied in [DM02]. Moreover, we determine which queries and transformations are feasible in polynomial time when knowledge is represented by PBC or CARD, and which are not (unconditionally or unless P = NP). In particular, the advantages and disadvantages compared to CNF are discussed.

The idea of this work was less to propose PBC and CARD as new languages useful for knowledge compilation—they are not because consistency checks for them are NP-complete—but to use the framework of the knowledge compilation map as a systematic way to better understand their properties. In fact, both cardinality and pseudo-Boolean constraints have been used in the field of pseudo-Boolean solving for a long time. However, some of their properties had not been well understood before our work. Since Romain's thesis project dealt with pseudo-Boolean constraints in general, the understanding of their basic properties that we gained through [23] gave valuable insights for the remainder of his work. In particular, it lead to the discovery of *irrelevant literals* [16] which were crucial for Romain's thesis and are discussed succinctly in Section 4.1.

A second small contribution to the knowledge compilation map is about so-called switch-lists. These are a representation language for Boolean functions introduced in [ČH17] as a generalization of interval representations [SGZ05]. The idea is to write the values of a Boolean function f on all lexicographically ordered inputs in a value table. Then, to encode f, it suffices to remember the value of f on the first input and the inputs at which the value of f changes from that of its predecessor. The resulting data structure is called a *switch-list* representation of f. Clearly, switch list representations can be far more succinct than value tables, e.g. for constant functions.

Čepek and Cromý [ČC20] analyzed switch-lists along the criteria of the knowledge compilation map and got a nearly complete picture. It turns out that switch-lists have many of the good properties of value tables while at the same time being generally much more succinct. The only questions [ČC20] leaves open is if switch-lists are closed under bounded disjunction and bounded conjunction, i.e., given two Boolean functions f_1 and f_2 represented by switch-lists, can one in polynomial time compute a switch-list representation of $f_1 \vee f_2$, resp. $f_1 \wedge f_2$. In [42] I show that this is not the case: there are Boolean functions f_1, f_2 such that any switch list representation of $f_1 \vee f_2$ is exponentially larger than those of f_1 and f_2. An analogous blow-up can also be shown for conjunction. This completes the analysis of switch-lists along the criteria of the knowledge compilation map.

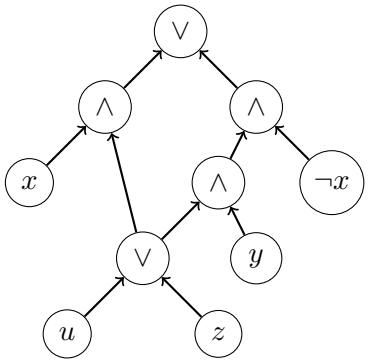


Figure 2.1.: A DNNF.

2.2. Background: Decomposable Negation Normal Form

Before continuing with more contributions in knowledge compilation, let us introduce some important data structures that will be central for essentially everything to come in the remainder of this thesis. The most important data structure in knowledge compilation are certainly *circuits in decomposable negation normal form* (DNNF) which were introduced by Darwiche in the groundbreaking work [Dar01a]. As the name suggests, a DNNF is a circuit over $\{\wedge, \vee, \neg\}$ in negation normal form (NNF), i.e., negation appear only in input gates. It is easy to see that, applying DeMorgan rules systematically, every Boolean circuit over $\{\wedge, \vee, \neg\}$ can be turned into NNF without much size increase, so this form alone is no restriction. The additional property that makes DNNF an interesting fragment of NNF is *decomposability*: all $\wedge$-gates have to be *decomposable* in the sense that the different subcircuits that feed into a $\wedge$-gate must all be on disjoint variables. See Figure 2.1 for an illustration. Note that concepts analogous to decomposability have been introduced as syntactic multilinearity in arithmetic circuit complexity, see e.g. [RY08, SY10], and for factorized representations in the context of databases [OZ15].

The main benefit of decomposability is that it makes consistency checks easy: given a Boolean function encoded as a DNNF, one can efficiently decide if it has a model, compute one if it exists and in fact even enumerate all models with polynomial delay [Dar01a, DM02]. Beyond this, DNNF generalize—and in fact are more succinct than—several well-known representations for Boolean functions like OBDD [Bry86] and DNF, see [DM02] for details. This combination of usefulness and succinctness makes DNNF arguably the central data structure in knowledge compilation. In Chapter 3, we will see many applications to different fields.

One additional useful property of DNNF is that one can define many useful fragments by putting restrictions on them. We will next introduce several such restrictions that will play crucial roles in the remainder of this thesis.

One canonical area where the representation languages introduced in knowledge compilation are applied is probabilistic reasoning. For example, one can translate, or *compile*, classifiers based on graphical models, e.g. Bayesian networks, into DNNF and then reason about the classifiers by querying the compiled representation [CD03]. In

this context, it is often crucial to efficiently count the (weighted) models of a Boolean function which for general DNNF is #P-hard. The underlying problem is that different inputs of $\vee$-gates may share models which makes counting the models for $\vee$-gates hard even if the model counts of their inputs are known; note that this difficulty already appears for DNF for which model-counting is well known to be #P-hard. To avoid this hardness result, Darwiche introduced the notion of *determinism* for DNNF which simply disallows this problematic behavior by requiring that for each $\vee$-gate the inputs have disjoint sets of models [Dar01b]. Weighted model counting for deterministic DNNF (short d-DNNF) is tractable which makes them well suited for probabilistic reasoning tasks [CD03, CKD13, SCD19]. Due to their importance, essentially all practical implementations of knowledge compilers create d-DNNFs or sub-classes thereof [Dar04, Dar11, MMBH12, CD13, OD15, LM17a].

It is often useful in applications of DNNF if one can compute conjunctions, disjunctions or negations of knowledge that is encoded by them; in knowledge compilation, this is called an *apply*-function. Unfortunately, for most fragments of DNNF such Boolean combinations generally blow up the representation size [DM02, 43]. One example of a class that allows an efficient apply-function are OBDD, assuming that the two OBDD one wants to combine have the same underlying variable order [Bry86]. This was generalized by Pipatsrisawat and Darwiche [PD08] to DNNF by imposing so-called *v-trees* to DNNF which play roughly the same role for DNNF as variable orders for OBDD: by definition, in DNNF at every $\wedge$-gate the variables below are split into two sets. A v-tree then is a tree-structure on the variables that prescribes *in which way* the variables have to be split. We spare the reader the technical details of the definition and just remark that when two Boolean functions are given as DNNF respecting the same v-tree, then one can in polynomial time compute a DNNF with the same v-tree computing their conjunction. This is the basis of so-called bottom-up compilation algorithms such as that for SDD [CD13], a restricted form of structured DNNF.

Finally, another property that is sometimes desirable in DNNF is smoothness: a DNNF is called *smooth* (sometimes also *complete*) if for every $\vee$-gate all inputs feeding into the gate have the same variables in their subtrees. See [SdBBA19] for more background and bounds.

2.3. Compiling

One of the major lines of practical knowledge compilation is compiling CNF-formulas into different representation languages, generally fragments of DNNF, see e.g. [Dar04, Dar11, MMBH12, CD13, OD15, LM17a] for work on implementations of this. However, as we will discuss in Section 2.4, this is generally not possible without an exponential blow-up in the representation size. Thus, a relevant theoretical question becomes for which classes of CNF-formulas, efficient compilation into DNNF is possible.

Work on this question concentrated mostly on the structure of such CNF-formulas as discussed in Chapter 1.1 for counting. The idea is again to assign a graph to every CNF-formula and then to determine classes of graphs that guarantee efficient compilation.

For example, it was known since the pioneering paper of Darwiche [Dar01a] introducing DNNF that if the treewidth of the so-called primal graph of the considered CNF-formulas is bounded, then there is a linear-time algorithm to compute DNNF representations.

Extending this result, I wrote a paper with Simone Bova, Florent Capelli, and Friedrich Slivovsky [33]. The underlying idea is that many of the graph width measures in Figure 1.1 generalize treewidth while still having some of its good algorithmic properties. We analyze these width measures for knowledge compilation and show that most of them allow efficient compilation algorithms. More precisely, we show that the traces of dynamic programming algorithms for #SAT can be used to construct structured deterministic DNNF representations of CNF-formulas. This is conceptually similar to the known fact that different subclasses of DNNF are essentially traces of DPLL-style algorithms [HD07], even thought the techniques are very different in our case. Our approach allowed us prove new upper bounds on the complexity of compiling CNF-formulas into structured deterministic DNNFs in terms of parameters such as the treewidth and the clique-width of the so-called incidence graph. In particular, we showed that every class of CNF-formulas of bounded clique-width has polynomial size structured deterministic DNNF representations. The degree of the corresponding polynomial is linear in the bound on the clique-width.

2.4. Lower Bounds for DNNF

One question of particular interest to me are lower bounds in knowledge compilation. It is not hard to see that under standard assumptions from complexity theory, no representation of Boolean functions that allows checking satisfiability efficiently can have small size, see e.g. [DM02]. Showing unconditional lower bounds on Boolean circuits is often far harder, even for relatively weak classes of circuits. In fact, this is a major line of complexity theory, see e.g. [Juk12] for a textbook presentation.

For the languages commonly studied in knowledge compilation, even a few years ago, lower bounds techniques were arguably not very developed. For various forms of binary decision diagrams like OBDDs and FBDDs there was a wealth of knowledge, see [Weg00]. For more general languages of the DNNF family, there was essentially one approach: for monotone functions, it is not hard to see that one can w.l.o.g. assume that all DNNF encodings are monotone as well, i.e., the DNNF representation contains no negative literals [43]. Then one can directly apply lower bounds on monotone circuits, see e.g. [Juk12], to get lower bounds for DNNF.

While this technique gives lower bounds for DNNF, it is unsatisfying in several respects: at a philosophical level, it is unsatisfying that one learns nearly nothing about DNNF themselves since the approach is exclusively using the properties of monotone circuits and lower bounds for them. More pragmatically, the approach by monotone circuit lower bounds is very coarse and there is no obvious way of extending it to understand the relationships between different sub-classes of DNNF, e.g. deterministic and/or structured DNNF. Also, since some interesting classes of functions for which one would like to have lower bounds in fact have small monotone DNFs, e.g. so-called lineages in the area of

probabilistic databases [JS13], the approach sketched above is inherently not useful there.

The only lower bound that I am aware of that explicitly uses the properties of DNNF are bounds for certain functions in the master's thesis of Krieger [Kri06]. However, the techniques in that work were very specifically tailored to the functions considered there and there is no apparent way of generalizing them to more general classes of functions.

2.4.1. The general framework

With Simone Bova, Florent Capelli, and Friedrich Slivovsky [27], we proposed a more general and flexible framework to show lower bounds for (subclasses of) DNNF and related languages that is based on so-called multi-partition communication complexity. I will first discuss the case of general DNNF in this section before showing several extensions and adaptions in the following sections.

The basic idea of the lower bound framework is is as follows: we show that every function f in variables X computed by a DNNF of size s can be written as

$$f(X) = \bigvee_{i \in [s]} f'_i(X'_i) \wedge f''_i(X''_i) \tag{2.1}$$

where for every i we have that (X'_i, X''_i) is a partition of X such that $\min(|X'_i|, |X''_i|) \geq |X|/3$. Functions of the form

$$f'_i(X'_i) \wedge f''_i(X''_i)$$

are called *combinatorial rectangles* and are studied extensively in communication complexity, a well-established subarea of complexity theory. Representations of Boolean functions in the form of (2.1) are called *rectangle covers*. Rectangle covers are a central object in communication complexity, but most of the time in that area one assumes that the partition (X'_i, X''_i) is the same for all rectangles in a cover. In this setting, one takes the variables X'_i and X''_i as the variables respectively known to two players who want to compute a function together which naturally leads to rectangle covers, see e.g. [KN97]. The setting for DNNF is more general than this usual setting from communication complexity and is called *multi-partition communication complexity* [DHJ+04].

With Equation 2.1, one can prove lower bounds for DNNF representations of functions: if one can show for function f that the number of disjuncts necessary in a rectangle cover, called the *size* of a rectangle cover, is at least s, then any DNNF representation of f must have size s as well. Thus one can apply known lower bounds on the size of rectangle covers from the literature to show lower bounds for DNNF representations. So in [27], besides proving the representation of Equation 2.1, we showed how to apply it by using results from [JS02, DHJ+04] to give DNNF lower bounds.

Since we did not know [JS02, DHJ+04] when first working in this direction, we also showed our own lower bounds on sizes of rectangle covers in the paper [43] which was written by the same authors. The functions we studied there were as follows: fix a graph $G = (V, E)$, then one defines

$$F_G := \bigwedge_{uv \in E} (x_u \vee x_v)$$

as a CNF-formula in the variables $\mathsf{var}(F_G) := \{x_v \mid v \in V\}$. We remark that the formulas of the form F_G were already used by Razgon [Raz16, Raz14] before to give lower bounds for certain (nondeterministic) decision diagrams. Later they have also been used in [ACMS20].

The intuition behind the formulas F_G is that if G is "sufficiently complicated", then F_G should be hard to represent. In [43], we showed that this is indeed the case: if G is a so-called *expander graph*, then any DNNF representation of F_G must be of size exponential in $|V(G)|$, see also Florent Capelli's thesis [Cap16] for an improved presentation of this result. [ACMS20] makes our result more quantitative by showing that for bounded degree graphs, the representation size of F_G is exponential in the treewidth of G (note that the corresponding upper bound is clear by the results of [Dar01a, 33]). We remark that these results of [ACMS20] crucially build on our results in [43, 27].

2.4.2. Deterministic DNNF

Remember that deterministic DNNF, short d-DNNF, are DNNF in which for all $\vee$-gates no two inputs share any models. It is now known that representations of knowledge in d-DNNF are generally large. From [DM02], it had been known under standard complexity theoretical assumptions that d-DNNF for certain function must be large [DM02]. In [27] we make this result unconditional by giving an explicit connection between d-DNNF lower bounds and communication complexity: if a function is represented by a d-DNNF of size s, one gets a representation as in Equation 2.1 with the additional property that all rectangles $f'_i(X'_i) \wedge f''_i(X''_i)$ are disjoint, i.e., whenever $i \neq j$, then $f'_i(X'_i) \wedge f''_i(X''_i)$ and $f'_j(X'_j) \wedge f''_j(X''_j)$ have no common models. We use this and a result by Sauerhoff on multi-partition communication complexity from [Sau03] to unconditionally show an exponential separation between general DNNF and their subclass d-DNNF, i.e., there are Boolean functions with polynomial-size DNNF representations such that all d-DNNF representations are of exponential size.

2.4.3. Structured DNNF

As discussed in Section 2.2, it is sometimes desirable to have, instead of general DNNF, structured DNNF, i.e., such that respect a v-tree, since this allows to perform more transformations on the encoded functions, in particular efficient conjunctions of two functions [PD08]. Interestingly, Pipatsrisawat and Darwiche show already in [PD10] that there are functions for which any structured DNNF representation is exponentially bigger than general DNNF. They do so by showing essentially a version of Equation 2.1, however with the crucial difference that all disjuncts have the *same* variable partition (X'_i, X''_i).

In [27], we show that this version of Equation 2.1 embeds seamlessly into the framework for DNNF lower bounds which allows us to explicitly make the connection to communication complexity. Since communication complexity in this setting is far easier and better understood than that for the multi-partition case, this allows us to use a wealth of techniques from the literature and in particular answer an open question from [PD10] rather easily.

2.5. Lower Bounds for Approximate Knowledge Compilation

As explained in the previous section, the tools of [27] allow showing strong lower bounds for d-DNNF representations. This is of course bad news for application areas such as probabilistic reasoning that use d-DNNF as a data structure to reason on: if already the compilation of the knowledge at hand into d-DNNF is infeasible, then the whole approach might already fail at this compilation step.

Fortunately, this bad news is not necessarily a fatal problem for probabilistic reasoning. Since graphical models like Bayesian networks are almost exclusively inferred by learning processes, they are inherently not exact representations of the world. Thus, when reasoning about them, in most cases the results do no have to be exact but approximate reasoning is sufficient, assuming that the approximation error can be controlled and is small. It is thus natural in this context to consider *approximate knowledge compilation*: the aim is no longer to represent knowledge exactly as one allows a small number of errors. Recently, Chubarian and Turán [CT20] have shown, building on [GKM$^+$11], that this approach is feasible in some settings: it is possible to compile approximations of so-called Tree Augmented Naive Bayes classifiers (TAN) (or more generally bounded pathwidth Bayes classifiers) into OBDDs efficiently. Note that efficient exact compilation is ruled out in this setting due to strong OBDD lower bounds for threshold functions from [TNY97] which imply lower bounds for TANs.[1]

In the paper [15] with my student Alexis de Colnet, we complement the positive results of [CT20] by extending lower bounds for exact representations to lower bounds for approximations. Similar questions had been treated before for OBDDs and some extensions such as *read-k branching programs*, see e.g. [KSW99, BSW02]. We extend this line of work in two ways: we show that the techniques used in [BSW02] can be adapted to show lower bounds for the approximation by d-DNNFs and prove that there are functions for which any d-DNNF computing a non-trivial approximation must have exponential size.

As a second contribution, we refine the approximation notion used in [BSW02] which we call *weak approximation*. For this notion, the approximation quality is measured as the probability of encountering an error when comparing a function and its approximation on a random input. It follows that all families of Boolean functions for which the probability of encountering a model on a random input is very small can be approximated trivially by constant functions. This makes weak approximation easy for rather uninteresting reasons for many functions, e.g. most functions given by CNF-formulas. Moreover, it makes the approximation quality sensitive to encodings, in particular the use of auxiliary variables that functionally depend on the input as these decrease the fraction of models with respect to all possible inputs. In general, the space of satisfying assignments is arguably badly described by weak approximations. In particular, the relative error for model counting and probability evaluation is unbounded which makes that notion useless for probabilistic reasoning.

[1] Note that the lower bound of [TNY97] for threshold functions has recently been extended from OBDD to the generally exponentially more succinct DNNF by my student Alexis de Colnet using completely different techniques [dC20].

We remedy the situation by formalizing a new notion of approximation for knowledge compilation which we call *strong approximation*. It is modeled to allow efficient counting with approximation guarantees and is insensitive to addition of functionally dependent auxiliary variables. While not formalized as such, it can be verified that the OBDDs of [CT20, GKM$^+$11] are in fact strong approximations in our sense. We then show that weak and strong approximations differ by exhibiting a family of functions that has trivial weak approximations but any d-DNNFs approximating it non-trivially must be of exponential size. The hard functions for which we show these lower bounds are linear codes used in [DHJ$^+$04]. To show our lower bounds we adapted a discrepancy based argument of [BSW02] to the framework of [27] sketched in Section 2.4.

We remark that approximation in knowledge compilation had been considered before—in fact one of the earliest lines of work in the setting was approximating Boolean functions by Horn formulas [SK96]. However, the focus was different in this setting: on the one hand, Horn formulas are not fully expressive so the question becomes that of understanding the formulas that are the best out of all Horn formulas approximating a function instead of requesting error guarantees for the approximation. On the other hand, that line of work was less concerned with the quality of the approximating formulas. Our work in [15] is different in these respects: since we deal with a fully expressive representation language, the main concern becomes that of a trade-off between the quality of approximation (measured in the number of inputs in which the function at hand and its approximation differ) and the representation size of the approximation.

2.6. Parameterized Lower Bounds

In [29], I used the results of [27] to complement the findings of [33] discussed in Section 2.3: remember that in [33] algorithms compiling CNF-formulas with restricted underlying graph structure were presented. This showed that popular graph width measures like treewidth and cliquewidth can be used in knowledge compilation. More specifically, every CNF-formula of incidence *treewidth* k and size n can be compiled into a DNNF of size $2^{O(k)}n$. Moreover, if k is the incidence *cliquewidth*, the size bound on the encoding becomes $n^{O(k)}$. As has long been observed, $2^{O(k)}n$ is of course far preferable to $n^{O(k)}$ for nontrivial sizes of n—in fact, this is the main premise of the field of parameterized complexity theory, see e.g. [FG06]. Consequently, the results of [33] leave open the question if the algorithm for clique-width based compilation of CNF-formulas can be improved.

In fact, the paper [33] already gives a partial answer to this question, proving that there is no compilation algorithm achieving fixed-parameter compilability, i.e., a size bound of $f(k)p(|F|)$ for a function f and a polynomial p. But unfortunately this result is based on the plausible but rather non-standard complexity assumption that not all problem in W[1] have FPT-size circuits. In [29] I show that this assumption is not necessary. I give a lower bound of $|F|^{\Omega(k)}$ for formulas of *modular incidence treewidth* k where modular treewidth is a restriction of cliquewidth proposed in [PSS13]. It follows that the result in [33] is essentially tight. Moreover, I give a lower bound of $|F|^{\Omega(\sqrt{k})}$ for formulas of

neighborhood diversity k [Lam10]. This intuitively shows that all graph width measures that are stable under adding modules, i.e., adding a new vertex that has exactly the same neighborhood as an existing vertex, behave qualitatively worse than treewidth for compilation into DNNFs.

To show these results, I use the linear codes of [DHJ$^+$04] which by [27] were already known to be hard for DNNF encodings. However, when using them directly as in [DHJ$^+$04], they are actually too hard since they cannot be expressed as CNF-formulas of bounded cliquewidth. Thus, one has to scale them to have the right complexity to allow showing DNNF lower bounds while also having the desired encodings into CNF. The main technical contribution of [29] is thus actually showing that the latter is possible: it is shown that $k \log(n)$ parity constraints, each on the same n variables, can be encoded by a polynomial size CNF-formula of cliquewidth k. The main idea for this is that, instead of encoding individual constraints, one can bundle them into groups of $\log(n)$ constraints each and then proceed by compiling those groups of constraints by small formulas of cliquewidth independent of k. Doing this for all groups then yields the desired encoding. This approach inspired a more general bundling technique in [3] where it is used to show that *any* CNF-formula F in n variables and of treewidth k can be rewritten into an encoding (with additional variables) of cliquewidth $O(\lceil k/log(n) \rceil)$, see section 3.3 for more on this work.

3. Applications of Knowledge Compilation

In this chapter, I will discuss applications of knowledge compilation, in particular DNNF based techniques, that I have proposed in several areas. One common point of many of these contributions is that a priori it is not clear that DNNF could be helpful in the respective settings at all, and certainly DNNF were not invented with most of these applications in mind.

3.1. Quantified Boolean Formulas

This section gives a survey of several of my works that are connected to quantified Boolean formulas in different ways, ranging from solving them, to applications for other problems, and to proof complexity.

3.1.1. Solving QBF with the Help of Knowledge Compilation

It is well known that restricting the interaction between variables and clauses in CNF-formulas makes several hard problems on them tractable. For example the propositional satisfiability problem SAT and its counting version #SAT can be solved in time $2^{O(k)}|F|$ when F is a CNF formula whose primal graph is of treewidth k [Sze04, SS10a]. As discussed in Section 1.1, many extensions of this result have been shown over the years for more general graph width measures [FMR08, PSS16, SS13, STV15]. In [19], a joint paper with Florent Capelli, we generalize the algorithm for treewidth in a different direction by considering decision and counting for quantified Boolean formulas (QBF) with a bounded number of quantifier alternations, i.e., we consider problems higher up in the polynomial hierarchy than SAT, resp. higher in the counting hierarchy than #SAT. It was already known before that QBF as well as projected model counting, i.e., model counting for QBF with free variables and one block of existentially quantified variables, are both fixed-parameter tractable parameterized by treewidth [Che04, FHMW18]. In [19] we generalize both these results by showing that counting the models of QBF with free variables is fixed-parameter tractable parameterized by treewidth for any bounded number of quantifier alternations. Moreover, the same is true for the strictly more general parameter of signed cliquewidth [FMR08].

Our approach to showing these results is completely different from those used before in the literature for treewidth restrictions of problems harder than the NP, resp. #P: we do not perform dynamic programming as e.g. in [Che04, FHMW18, DSW10, BOW16]. Instead, we encode all models of the underlying CNF-formula of the given QBF into complete structured d-DNNF [PD08]. Afterwards, we perform quantifier elimination on this representation. When all quantifiers are eliminated, we can answer the query on the input QBF by standard algorithms for d-DNNF.

One crucial advantage of our approach is that the first step, the compilation into d-DNNF, was already solved before: as discussed in Section 2.3, there are several compilation algorithms for structurally restricted CNF-formulas [Dar01a, 33]. Thus we can take these algorithms as a black box and get the compiled representations for free without doing any additional dynamic programming.

It thus only remains to eliminate quantifiers on d-DNNF. Unfortunately, there are unconditional, exponential lower bounds showing that in general quantifier elimination on d-DNNF is impossible without blowing up the size of the representation [PD10]. We avoid this problem by identifying a notion of *width* for complete structured d-DNNF that is modeled after the classical width of complete OBDD. We go on to show that the size explosion during the quantifier elimination is in fact not in the size of the input but only in its width by giving a relatively simple algorithm inspired by determinization of finite automata. Since several of the compilation algorithms mentioned above, in particular that of [33][1], yield d-DNNF whose width is independent of the input size, we get an algorithm for several restricted classes of QBF.

The resulting algorithm can be used to show that the number of models of a partially quantified CNF-formula F of treewidth k with t blocks of quantifiers can be computed in time $2^{\cdot^{\cdot^{\cdot^{2^{O(k)}}}}}|F|$ with $t+1$ exponentiations. This generalizes the result of [Che04] where the fixed-parameter tractability of QBF on such formulas was shown with a comparable complexity. Note that the tower of exponentials in the runtime is unavoidable under standard complexity assumptions [PV06, FHP20]. Our algorithm generalizes the recent result of [FHMW18] on model counting in the presence of a single existential variable block. Finally, it also applies to the more general notions of incidence treewidth and signed cliquewidth.

Note that the notion of width of a complete structured DNNF and some of the related results in [19] turn out to be useful in other settings: they were used in [ACMS20] to tighten some of the results of the prior conference version of that paper [AMS18]. Moreover, width plays a crucial role in my work that will be presented in Section 3.3.

3.1.2. QBF as an Alternative to Courcelle's Theorem

Courcelle's seminal theorem [Cou90] states that every graph property definable in monadic second-order logic can be decided in linear time on graphs of constant treewidth. While the statement of this theorem might sound abstract to the unsuspecting reader, the consequences are tremendous: since a huge number of computational problems can be encoded in monadic second-order logic, this gives automatic linear-time algorithms for a wealth of problems in such diverse fields as combinatorial algorithms, artificial intelligence and databases; out of the plethora of such papers let us only cite [GPW10, Dun07] that treat problems that will reappear in this section. Courcelle's Theorem is one of the cornerstones of the field of parameterized algorithms, see e.g. [FG06].

Unfortunately, its strength comes with a price: while the runtime dependence on the

[1] In fact, since we had not defined width of structured complete DNNF yet when writing [33], this is only implicit there. We made it explicit in the journal version that is currently under submission.

size of the problem instance is linear, the dependence on the treewidth is unclear when using this approach. Moreover, despite some progress (see e.g. the survey [LRRS14]) Courcelle's Theorem is largely considered impractical due to the gigantic constants involved in the construction. Since generally these constants are unavoidable [FG04], showing linear time algorithms with Courcelle's Theorem can hardly be considered as a satisfying solution.

As a consequence, linear time algorithms conceived with the help of Courcelle's Theorem are sometimes followed up with more concrete algorithms with more explicit runtime guarantees, often by dynamic programming or applications of a datalog approach [DPW12, GPW10, JPRW08]. Unfortunately, these hand-written algorithms tend to be very technical, in particular for decision problems outside of NP. Furthermore, even this meticulous analysis sometimes gives algorithms with a dependence on treewidth that is a tower of exponentials.

In [22], together with Michael Lampis and Valia Mitsou, we propose a QBF-approach in the setting. More concretely, we propose reductions to QBF combined with the use of QBF algorithms such at that by Chen [Che04] or that discussed in Section 3.1.1 as a simple way of constructing linear-time algorithms for problems beyond NP parameterized by treewidth. In particular, we use the proposed method in order to construct (alternative) algorithms for a variety of problems stemming from artificial intelligence: abduction, circumscription, abstract argumentation and the computation of minimal unsatisfiable sets in unsatisfiable formulas. The advantage of this approach over Courcelle's Theorem or tedious dynamic programming is that the algorithms we provide are almost straightforward to produce, while giving bounds on the treewidth that asymptotically match those of careful dynamic programming. We also show that our algorithms are asymptotically best possible, giving matching complexity lower bounds.

Our algorithmic approach might at first sight seem surprising: since QBF with a fixed number of alternations is complete for the different levels of the polynomial hierarchy, there are trivially reductions from all problems in that hierarchy to the corresponding QBF problem. So what is new about this approach? The crucial observation is that in general reductions to QBF guaranteed by completeness do not maintain the treewidth of the problem. Moreover, while the QBF algorithms run in linear time, there is no reason for the reduction to QBF to run in linear time which would result in an algorithm with overall non-linear runtime.

The runtime bounds that we give are mostly of the form $2^{2^{O(k)}}n$ where k is the treewidth and n the size of the input. To complement these results, starting from lower bounds for QBF [LM17b], we also show that these runtime bounds are essentially tight as there are no algorithms with runtime $2^{2^{o(k)}}2^{o(n)}$ for the considered problems. Our lower bounds are based on the *Exponential Time Hypothesis (ETH)* which posits that there is no algorithm for 3SAT with runtime $2^{o(n)}$ where n is the number of variables in the input. ETH is by now widely accepted as a standard assumption in the fields of exact and parameterized algorithms for showing tight lower bounds, see e.g., the survey [LMS11]. We remark that our bounds confirm the observation already made in [MM16] that problems complete for the second level of the polynomial hierarchy parameterized by treewidth tend to have double-exponential runtime in the treewidth.

In conclusion, [22] shows that reductions to QBF can be used as a simple technique to show algorithms with essentially optimal runtime for a wide range of problems.

3.1.3. Proof Complexity of Symbolic QBF Solving

Unlike in practical SAT solving, which is dominated by Conflict-Driven Clause Learning (CDCL), in QBF solving there is no single approach that is clearly dominant in practice. Instead, modern solvers are based on variety of techniques, such as (quantified) CDCL [ZM02, LB10, PSS19], expansion of universal variables [Bie04, JKMC16, BBH$^+$18], and abstraction [RT15, JM15, Ten16].

In practice, these techniques turn out to be complementary, each having strengths and weaknesses on different classes of instances [PT09, HPSS18, LE18]. This complementarity of solvers can be analyzed theoretically by considering proof complexity. Essentially, the different paradigms used in solvers can be formalized as proof systems for QBF, which then can be analyzed with mathematical methods. Then, by separating the strength of different proof systems, one can show that the corresponding solvers are unable to solve problems efficiently that can be dealt with by other solvers. This motivation has led to a great interest in QBF proof complexity over the last few years and resulted in a good understanding of common QBF proof systems and how they relate to each other, see [BCJ19, BBCP20] and the references therein.

In a joint paper with Friedrich Slivovsky [14], we focus on a *symbolic* approach to QBF solving that was originally proposed by Pan and Vardi and implemented in the QBDD system [PV04]. Its underlying idea is to use OBDDs to represent constraints inside the solver, instead of clauses as used by most other SAT and QBF solvers. In [14], we formalize QBDD as a proof system in which the lines are OBDDs. More specifically, we consider QBF proof systems that are obtained from propositional OBDD-proof systems by adding $\forall$-reduction, a standard way of eliminating universally quantified variables in QBF reasoning (cf. [BBCP20]). Propositional proof systems using OBDDs as lines have been studied intensively since the introduction of this model in [AKV04], see e.g. [BIKS18]. We thus consider lifting these systems to QBF by adding $\forall$-reduction as very natural.

Analyzing the strength of OBDD-refutations, we first show that, even for a weak propositional system that allows only conjunction of previously inferred OBDDs and forgetting of variables, the resulting QBF proof system, which we refer to as OBDD$(\wedge, \exists, \forall)$ and which corresponds to traces of the QBDD-system, simulates QU-resolution, a standard QBF proof system, with only polynomial overhead. We also show that OBDD$(\wedge, \exists, \forall)$, and in fact also QBDD, can make use of structural properties of QBF in the sense that instances of bounded pathwidth and bounded quantifier alternation can be solved efficiently. We do this by using a result on variable elimination for OBDDs from [19], see Section 3.1.1, to show that the intermediate OBDDs in QBDD are not too big in this setting. We then observe that other QBF proof systems from the literature have hard instances of bounded pathwidth and bounded quantifier alternation. This shows that OBDD$(\wedge, \exists, \forall)$ can efficiently refute QBFs that are out of reach for many other well-known systems. It follows that, at least in principle, QBDD can solve instances that other, more modern solvers cannot.

The main technical contribution of [14] is a lower bound technique for OBDD-refutations. We consider the strongest possible propositional system, which is semantic entailment of OBDDs. Using an approach of [BJ12, BBCP20] called *strategy extraction* and a communication complexity result from [IW10] we reduce the question of showing lower bounds for OBDD-refutations to showing that there are functions f that do not have large monochromatic rectangles, i.e., any rectangle R with sufficiently many models must have models that are also models of f and models that are non-models of f. To the best of our knowledge, such bounds are only known for fixed variable partitions, so when the variables are split in a prescribed way. The canonical example for this is the so-called inner product function, see [KN97]. To prove lower bounds for OBDD-refutations that are independent of the variable order chosen for the OBDDs, we lift these classical bounds on the inner product function to a graph-based generalization which we show has essentially the same properties as the inner product function, but for *all* variable partitions.

3.2. Characterizing Tseitin-formulas with short regular resolution refutations

Resolution is one of the most studied propositional proof systems in proof complexity due to its naturality and it connections to practical SAT solving [Nor15, BN21]. A refutation of a CNF-formula in this system (a resolution refutation) relies uniquely on clausal resolution: in a refutation, clauses are iteratively derived by resolutions on clauses from the formula or previously inferred clauses, until reaching the empty clause indicating unsatisfiability. In [13], a joint paper with my student Alexis de Colnet, we consider regular resolution which is the restriction of resolution to proofs in which, intuitively, variables which have been resolved away from a clause cannot be reintroduced later on by additional resolution steps. This fragment of resolution is known to generally require exponentially longer refutations than general resolution [Goe93, AJPU07, Urq11, VEJN20] but is still interesting since it corresponds to DPLL-style algorithms [DLL62, DP60]. Consequently, there is quite some work on regular resolution, see e.g. [ABdR$^+$18, Urq87, BI13, BBI12] for a very small sample.

Tseitin-formulas are encodings of certain systems of linear equations whose structure is given by a graph [Tse68]. They have been studied extensively in proof complexity essentially since the creation of the field because they are hard instances in many settings, see e.g. [Urq87, Ben02, IO13, IRSS19, BBI12]. It is known that different properties of the underlying graph characterize different parameters of their resolution refutations [GTT20, AR11, IO13]. In [13], we extend this line of work by showing that treewidth determines the length of regular resolution refutations of Tseitin-formulas: classes of Tseitin-formulas of bounded degree have polynomial-length regular resolution refutations if and only if the treewidth of the underlying graphs is bounded logarithmically in their size. The upper bound for this result was already known from [AR11] where it is shown that, for every graph G, unsatisfiable Tseitin-formulas with the underlying graph G have regular resolution refutations of length at most $2^{O(\mathsf{tw}(G))}|V(G)|^c$ where c is a constant. We provide a matching lower bound:

Theorem. *Let $T(G, c)$ be an unsatisfiable Tseitin-formula where G is a connected graph with maximum degree at most Δ. The length of the smallest regular resolution refutation of $T(G, c)$ is at least $2^{\Omega(\mathsf{tw}(G)/\Delta)}|V(G)|^{-1}$.*

There were already known lower bounds for the length of resolution refutations of Tseitin-formulas based on treewidth before. For *general* resolution, a $2^{\Omega(\mathsf{tw}(G)^2)/|V(G)|}$ lower bound can be inferred with the classical width-length relation of [Ben02] and width bounds of [GTT20]. This gives a tight $2^{\Omega(\mathsf{tw}(G))}$ bound when the treewidth of G is linear in its number of vertices. For smaller treewidth, there are also bounds from [GIRS19] for the stronger proof system of depth-d Frege proofs which for resolution translate to bounds of size $2^{\mathsf{tw}(G)^{\Omega(1)}}$, but since the top exponent is significantly less than 1, these results are incomparable to ours. Better bounds of $2^{\Omega(\mathsf{tw}(G))/\log|V(G)|}$ for regular resolution that almost match the upper bound were shown in [IRSS19] for *regular* resolution refutations. Building on [IRSS19], we eliminate the division by $\log|V(G)|$ in the exponent and thus give a tight $2^{\Theta(\mathsf{tw}(G))}$ dependence.

As in [IRSS19], our proof strategy follows two steps. First, we show that the problem of bounding the length of regular resolution refutations of an *unsatisfiable* Tseitin-formula can be reduced to lower bounding the size of certain representations of a *satisfiable* Tseitin-formula. Itsykson et al. in [IRSS19] used a similar reduction of lower bounds for regular resolution refutations to bounds on read-once branching programs (1-BP) for satisfiable Tseitin-formulas, using the classical connection between regular resolution and the search problem which, given an unsatisfiable CNF-formula and a truth assignment, returns a clause of the formula it falsifies [LNNW95]. Itsykson et al. showed that there is a transformation of a 1-BP solving the search problem for an unsatisfiable Tseitin-formula into a 1-BP of pseudopolynomial size computing a satisfiable Tseitin-formula with the same underlying graph. This yields lower bounds for regular resolution from lower bounds for 1-BP computing satisfiable Tseitin-formulas which [IRSS19] also shows. Our crucial insight in [13] is that when more succinct representations are used to present the satisfiable formula, the transformation from the unsatisfiable instance can be changed to have only a polynomial instead of pseudopolynomial size increase. Concretely, the representations we use are DNNF which generalize 1-BP. We show that every refutation of an unsatisfiable Tseitin-formula can be transformed into a DNNF-representation of a satisfiable Tseitin-formula with the same underlying graph with only polynomial overhead.

In a second step, we then show for every satisfiable Tseitin-formula with an underlying graph G a lower bound of $2^{\Omega(\mathsf{tw}(G))}$ on the size of any DNNF computing the formula. To this end, we adapt the rectangle based techniques developed in [27] and presented in Section 2.4.1 to a parameterized setting. Our refinement takes the form of a two-player game in which the first player tries to cover the models of a function with few rectangles while the second player hinders this construction by adversarially choosing the variable partitions respected by the rectangles from a certain set of partitions. We show that this game gives lower bounds for DNNF, and consequently the aim is to show that the adversarial player can always force $2^{\Omega(\mathsf{tw}(G))}$ rectangles in the game when playing on a Tseitin-formula with graph G. This is done by proving that any rectangle for a

carefully chosen variable partition *splits* parity constraints of the formula in a way that bounds by a function of $\mathsf{tw}(G)$ the number of models that can be covered. We show that, depending on the treewidth of G, the adversarial player can choose a partition to limit the number of models of every rectangle constructed in the game to the point that at least $2^{\Omega(\mathsf{tw}(G))}$ of them will be needed to cover all models of the Tseitin-formula. As a consequence, we get the desired lower bound of $2^{\Omega(\mathsf{tw}(G))}|V(G)|^{-1}$ for regular resolution refutations of Tseitin-formulas.

3.3. Revisiting Graph Width Measures for CNF-Encodings

The work of [3] which is joint with my student Romain Wallon, is complementary to the algorithmic results discussed in Section 2.3 and Sections 3.1.1 and 3.1.2. As already seen there, graph width measures like treewidth and cliquewidth have been studied extensively in the context of propositional satisfiability. The general idea is to assign graphs to CNF-formulas and compute their width with respect to different width measures. Then, if the resulting width is small, there are algorithms that solve SAT, but also more complex problems like #SAT or MAX-SAT or even QBF efficiently, see e.g. [SS10a, FMR08, SS13, PSS16, STV15, Che04]. There is also a considerable body of work on reasoning problems from artificial intelligence restricted to knowledge encoded by CNF-formulas with restricted underlying graphs: for example, treewidth restrictions have been studied for abduction, closed world reasoning, circumscription, disjunctive logic programming [GPW10] and answer set programming [JPW09]. There is thus by now a large body of work on how problems can be solved on bounded width CNF-formulas for different graph width measures.

Curiously, however, there seems to be very little work on the natural question of what we can actually encode with these restricted CNF-formulas. This question is pertinent because of course good algorithms for problems are less attractive if they cannot deal with interesting instances. In [3], we make two main contributions on the expressiveness of bounded width CNF-formulas.

As a first main contribution, we show, for a wide class of width measures, that one can give width lower bounds of any encoding of a function by means of communication complexity. Such lower bounds were known for treewidth [BKM11], but with our general approach, we extend them for many different width measures, in particular (signed and unsigned) cliquewidth [FMR08, SS13], modular treewidth [PSS16] and mim-width [STV15]. As a consequence, in a sense, for all these measures, formulas of bounded width can only encode simple functions.

All these lower bounds not only work for *representations* of functions as CNF-formulas but also on *clausal encodings*, i.e., CNF-formulas using auxiliary variables. It is folklore that adding auxiliary variables can decrease the size of an encoding: for example the parity function has no subexponential CNF-representations but there is an easy linear size encoding using auxiliary variables. We observe a similar effect for the example of treewidth: we show that any CNF-representation of the $\mathsf{AtMostOne}_n$-function of n inputs without auxiliary variables has primal treewidth $n-1$ which is the highest possible.

But when authorizing the use of auxiliary variables, $\mathsf{AtMostOne}_n$ can be computed with formulas of bounded treewidth easily. This shows that lower bounds for clausal encodings are far stronger than those of CNF-representations. Considering that $\mathsf{AtMostOne}_n$ is arguably a very easy function, we feel that encodings with auxiliary variables are the more interesting notion in our setting so we focus on them in [3].

As we have seen before, this is of course not the first time that communication complexity has been used to show lower bounds on the size or width of representations for Boolean functions. In fact, this is one of the motivations for the development of the area and there is a large literature on this, see e.g. [KN97, Hro97, Juk12]. In particular, there are many results for showing lower bounds on different forms of branching programs by means of communication complexity, see e.g. [Weg00, DHJ$^+$04]. More recently, as we have seen in Section 2.4, this approach has been generalized to more general languages, in particular DNNF [PD10, 27]. However, beyond a lower bound on treewidth already shown in [BKM11], we are not aware of any use of communication complexity to prove bounds on width measures of CNF-formulas prior to our work.

In a second main contribution, we consider the *relative* expressive power of different graph width measures for clausal encodings. For the graph width measures studied in the literature, it is known that without auxiliary variables the expressiveness of bounded width CNF-formulas is different for all notions and they form a partial order with so-called mim-width as the most general notion, see e.g. [34, Section 5] and Figure 1.1 in Section 1.1. Somewhat surprisingly, the situation changes completely when one allows auxiliary variables: in this setting, the commonly considered width notions are all up to constant factors equivalent to either primal treewidth or to incidence cliquewidth. This is true for every individual function. We remark that for the parameters primal treewidth, dual treewidth and incidence treewidth, it was already known that the width of encodings minimizing the respective width measures differs only by constant factors [SS10b, BKM11, 22]. All other relationships are new.

We also show that, assuming that an optimal encoding of a function has at least primal treewidth $\log(n)$ where n is the number of variables, incidence cliquewidth and primal treewidth differ exactly by a factor of $\Theta(\log(n))$ for optimal encodings. So, up to a logarithmic scaling, in fact all the commonly used width measures coincide when allowing auxiliary variables. Note that this scaling exactly corresponds to the runtime differences of many algorithms: while treewidth-based algorithms often have runtimes of the form $2^{O(k)}n^c$ for treewidth k and a constant c, cliquewidth-based algorithms typically give runtimes roughly $n^{O(k')}$ for cliquewidth k'. These runtimes coincide exactly when treewidth and cliquewidth differ by a logarithmic factor which, as we show here, they do generally for encodings with auxiliary variables.

We finally use our main results for several applications. In particular, we answer an open question of [BKM11] on the cliquewidth of the permutation function PERM_n and generalize a classical theorem on planar circuits [LT80].

Most of our results use machinery for knowledge compilation that was presented in previous sections. In particular, we use a combination of the algorithm proposed in [33] (Section 2.3), the width notion for DNNF developed in [19] (Section 3.1.1) and the lower bound techniques introduced in [PD10] and [27] (Section 2.4). Relying on these building

blocks and combining them in the right way, most of our proofs become rather simple.

3.4. Succinctness Results for Arithmetic Circuits

Arithmetic circuits (AC) are a circuit model for representing polynomials by giving the order in which their inputs have to be combined by sums and multiplications. Thus, AC are not only very natural representations for real-valued polynomials, but also give programs for computing them; this can e.g. be traced back to [Val80] who called them $(+,\times)$-programs. Today AC play an important role in artificial intelligence because they encompass several classes of circuits with practical applications in probabilistic reasoning, for instance probabilistic sentential decision diagrams (PSDD) [KdBCD14] or sum product networks (SPN) with indicator variables [PD11]. AC are also strongly related to concepts such as AND/OR-circuits [DM07] and Cutset Networks [RKG14]. When used in probabilistic reasoning, AC always represent non-negative functions and are therefore called (somewhat misleadingly perhaps) *positive AC*. Positive AC constitute a subclass of what in the probabilistic graphical models community is called *probabilistic circuits* [CVVdB20]. In the literature, positivity is is often syntactically enforced by assuming that all constants in the computation are non-negative, see e.g. [Dar03, PD11], in which case the AC are called *monotone*. Essentially, compared to their monotone counterparts, positive AC encode programs which allow subtraction as an additional operation. This has no impact on the tractability of most operations performed on the AC [Den16] and it is known already since [Val80] that it can decrease the size of AC exponentially.

While research on arithmetic circuits in complexity theory focuses almost exclusively on trying to show lower bounds on the size of AC representing notoriously challenging polynomials like the permanent, see e.g. [JS82, SY10, Raz09], the goals pursued in artificial intelligence are often different: on the one hand, algorithms for generating AC from other models like Bayesian networks [CD08, CKD13, KdBCD14], or by learning from data [LD08, RL16], are a major focus. On the other hand, it is studied how imposing constraints on the structure of AC can render operations like computation of marginals or of maximum a posteriori hypotheses (MAP) or more complex queries tractable on them [HCD06, VCL$^+$21, KCL$^+$19]. In this latter line of work, properties analogous to those of subclasses of DNNF discussed in Section 2.2 are considered: decomposability (also called syntactic multilinearity), smoothness (also called completeness), and determinism. There is a similar trade-off as for classes of DNNF: on the one hand, more restrictive properties allow new operations, for example *structured decomposability* [KdBCD14, DVVdB20] is a rather restrictive notion considered and corresponds exactly to structuredness of DNNF. On the other hand, more general properties are sufficient to ensure tractability of few important operations. For instance *weak* decomposability (also called consistency) is a relaxation of decomposability which, if combined with smoothness, allows efficient marginals computation [PTPD15].

As in the setting of DNNF, the analysis of more restrictive properties is driven by the prospect of AC to support more operations efficiently and therefore be more useful

in practice. The quest for more generic properties is motivated by the succinctness of resulting AC: while generally all classes of AC commonly considered can represent all functions, more general classes should intuitively allow smaller representations. However, there has been little work trying to show that this intuition is correct by giving lower bounds. In complexity theory, there is a large amount of research on lower bounds focused on classes of AC with properties such as bounded-depth, tree-like structure, or multilinearity [GK98, Raz09, Raz10, SY10] that have deep implications in theory but are not particularly desirable in practice—with the exception of *syntactic* multilinearity which is in fact decomposability. In comparison to Boolean circuits, the succinctness analysis for classes of arithmetic circuits of practical interest is fairly young and far from complete [MM14, CD17].

In [12], with my student Alexis de Colnet, we initiate a systematic succinctness map for AC modeled after that proposed in [DM02] for NNF. We focus on classes of AC with 0/1-variables that respect decomposability or weak decomposability and possibly determinism and/or smoothness. Most of our results deal with classes of *monotone* AC and are obtained by lifting results from the existing succinctness map for NNF. To this end, we observe that understanding the succinctness relations between different classes of monotone AC reduces to understanding that between classes of NNF with analogous restrictions. However, several classes of NNF obtained with the reduction, namely those respecting weak decomposability, have only recently been introduced [AAC$^+$19] and thus their position in the maps has not been studied. To analyze monotone AC, we thus prove the missing succinctness relations for these classes. From the map for NNF and the lifting technique, we obtain the complete map linking the eight classes of *monotone* AC one gets combining the different restrictions we consider. In a modest contribution to the understanding of *positive* AC, we show that under particular restrictions, all including determinism, the expressive power of classes of positive AC coincide with that of their monotone counterparts. Thus some succinctness relations in the monotone map easily extend to the positive map. However, for positive AC, several relations between classes remain open. Finally, in an effort to motivate further research on the succinctness relations left to prove, we describe a technique to show lower bounds on the size of positive AC. We apply it to prove lower bounds for positive AC with *structured* decomposability, which is the case for e.g. PSDD [KdBCD14].

3.5. Enumeration in Database Theory

When a computational problem has a great number of solutions, computing all of them at once can take an unreasonable amount of time. *Enumeration algorithms* are an answer to this challenge, and have been studied in many contexts (see [Was16] for an overview). Such algorithms generally consist of two phases: first, in a *preprocessing phase* the input is preprocessed. Second, using the results of the preprocessing, in an *enumeration phase* the solutions are computed one after the other, while limiting the amount of time between each pair of successive solutions, called the *delay*.

I have several publications on such enumeration algorithms in which I focused on a

well-studied class of efficient enumeration algorithms with very strict requirements: the preprocessing must be *linear* in the input data, and the delay between successive solutions must be *constant*. Such algorithms have been studied in particular in database theory, to enumerate query answers (see e.g. [DG07, Bag06, DSS14, BDFG10, BDG07, KS13b, KS13a, OZ15, CK19, CK18, BKS18, SSV18, BKS17b, BKS17b] for some samples and the surveys [Seg14, BGS20]).

3.5.1. An Approach Based on DNNF

One shortcoming of most existing enumeration algorithms is that they are typically shown by constructing a custom index structure tailored to the specific problem, and designing custom preprocessing and enumeration algorithms. This makes it difficult to generalize these results to other problems, or to implement them efficiently. It would thus be far preferable if enumeration for multiple problems could be performed using one generic representation of the results to enumerate, reusing algorithms for the preprocessing and enumeration phases. Accordingly, the paper [25] which is joint work with Antoine Amarilli, Pierre Bourhis and Louis Jachiet, proposes a new framework for constant-delay enumeration, inspired by knowledge compilation. Using knowledge compilation to succinctly represent data to be enumerated is very natural: after all, the area studies how the solutions to computational problems can be compiled to generic representations on which reasoning tasks can then be solved using general-purpose algorithms. We show how this approach can be implemented for constant-delay enumeration. At this point in this thesis, it might be unsurprising for the reader that the data structure from knowledge compilation that we use are DNNF, more precisely structured d-DNNF.

Our main technical contribution in [25] is an efficient algorithm for enumerating the satisfying *valuations* of a structured d-DNNF where a valuation is an encoding of an assignment by giving the positions of the 1-entries. Note that crucially this encoding can be far more succinct than giving the whole assignment in the case where the Hamming weight of an assignment is low, which is the case in the setting we consider. For our algorithm we assume that a v-tree [PD08] of the input d-DNNF is given as part of the input. This v-tree can easily be computed in most applications.

Our first main theorem shows that we can enumerate the satisfying valuations of a structured d-DNNF with linear preprocessing and delay linear in the Hamming weight of each valuation. Further, our second main theorem shows that, if we impose a constant bound on the Hamming weight, we can enumerate the valuations with constant delay.

Our results are shown for d-DNNF under a semantics where negation is implicit, i.e., variables that are not tested must be set to zero. In analogy to zero-suppressed OBDD [Weg00], we call this semantics *zero-suppressed*. The preprocessing phase of our algorithm rewrites such circuits to a normal form and pre-computes a multitree reachability index on them, which allows us to enumerate efficiently the *traces* of the circuit, yielding the desired valuations. To enumerate for d-DNNF in standard semantics, we show how to rewrite the input circuit to zero-suppressed semantics, using structuredness, and a new notion of *range gates* to make the process efficient. The overall proof is very modular.

The second contribution of [25] is giving evidence that our circuit-based framework

and enumeration results are useful in database theory. As a proof of concept, we present two known results that we can extend, or recapture with independent proofs: First, we re-prove with our framework that the answers to MSO queries on trees and bounded treewidth structures can be enumerated with linear preprocessing and delay linear in each assignment, i.e., constant-delay if the free variables are first-order. This was previously shown by Bagan [Bag06] with a custom construction, and by Kazana and Segoufin [KS13b] using a powerful result of Colcombet [Col07]. Our proof follows our proposed approach: we compute a circuit representation of the output following the provenance constructions in [ABS15], and simply apply our enumeration result to this circuit. Second, we show how our approach by d-DNNF generalizes an enumeration algorithm for so-called deterministic factorized representations [OZ15], making it possible to efficiently enumerate for larger classes of such representations.

3.5.2. Enumeration Under Updates

In a follow-up paper to [25], we show how database updates can be efficiently incorporated into certain enumeration tasks in [20] which is joint work with Antoine Amarilli and Pierre Bourhis. Concretely, we revisit how to evaluate MSO queries with free variables on trees, within the framework of enumeration algorithms but this time allowing certain changes to the input. As discussed above, previous work had shown how to enumerate answers with linear-time preprocessing and delay linear in the size of each output, i.e., constant-delay for free first-order variables. In [20], we extend this result to support relabelings, a restricted kind of update operations on trees which allows us to change the node labels. Our main result shows that we can enumerate the answers of MSO queries on trees with linear-time preprocessing and delay linear in each answer, while supporting node relabelings in logarithmic time. To prove this, we reuse the circuit-based enumeration structure from [25], and develop techniques to maintain its index under node relabelings. We also show how enumeration under relabelings can be applied to evaluate practical query languages, such as aggregate, group-by, and parameterized queries.

3.5.3. Enumeration for Document Spanners

Information extraction from text documents is an important problem in data management. One approach to this task has recently attracted a lot of attention: it uses *document spanners*, a declarative logic-based approach first implemented by IBM in their tool SystemT [Res18] and whose core semantics has then been formalized in [FKRV15]. The spanner approach uses variants of regular expressions (e.g. *regex-formulas* with variables), compiles them to variants of finite automata (e.g., *variable-set automata*, for short *VAs*), and evaluates them on the input document to extract the data of interest. After this extraction phase, algebraic operations like joins, unions and projections can be performed. The formalization of the spanner framework in [FKRV15] has led to a thorough investigation of its properties by the theoretical database community, see e.g. [Fre17, FKP18, MRV18, FH18, FRU$^+$18, PFKK19, Pet21, FT20, SS21].

In a joint paper with Antoine Amarilli, Pierre Bourhis and Matthias Niewerth [20], we consider the basic task in the spanner framework of efficiently computing the results

of the extraction, i.e., computing without duplicates all tuples of ranges of the input document (called *mappings*) that satisfy the conditions described by a VA. As many algebraic operations can also be compiled into VAs [FKP18], this task actually solves the whole data extraction problem for so-called *regular spanners* [FKRV15]. While the extraction task is intractable for general VAs [Fre17], it is known to be tractable if we impose that the VA is *sequential* [FKP18, FRU$^+$18], which requires that all accepting runs describe a well-formed mapping. Even then, however, it may still be unreasonable in practice to materialize all mappings: if there are k variables to extract, then mappings are k-tuples and there may be up to n^k mappings on an input document of size n, which is unrealistic if n is large. For this reason, recent works [MRV18, FRU$^+$18, FKP18] have studied the extraction task in the setting of *enumeration algorithms* as discussed above. Specifically, [FKP18] has shown how to enumerate the mappings with delay linear in the input document and quadratic in the VA, i.e., given a document d and a functional VA A (a subclass of sequential VAs), the delay is $O(|A|^2 \times |d|)$.

Although this result ensures tractability in both the size of the input document and the automaton, the delay may still be long as $|d|$ is generally very large. In [FRU$^+$18] it has been shown that constant delay could be achieved when enumerating the mappings of VAs if we only focus on data complexity, i.e., for any *fixed* VA, we can enumerate its mappings with linear preprocessing and constant delay in the input document. However, the preprocessing and delay in [FRU$^+$18] are exponential in the VA because they first determinize it [FRU$^+$18]. This is problematic because the VAs constructed from regex-formulas [FKRV15] are generally nondeterministic.

Thus, to efficiently enumerate the results of the extraction, we would ideally want to have the best of both worlds: ensure that the *combined complexity* (in the sequential VA and in the document) remains polynomial, while ensuring that the *data complexity* (in the document) is as small as possible, i.e., linear time for the preprocessing phase and constant time for the delay of the enumeration phase. However, up until [20], there was no known algorithm to satisfy these requirements while working on nondeterministic sequential VAs. Further, it was conjectured that such an algorithm is unlikely to exist [FRU$^+$18] because the related task of *counting* the number of mappings is hard for such VAs.

In [20], we show that nondeterminism is in fact not an obstacle to enumerating the results of document spanners: we present an algorithm that enumerates the mappings of a nondeterministic sequential VA in polynomial combined complexity while ensuring linear preprocessing and constant delay in the input document. This answers the open question of [FRU$^+$18], and improves on the bounds of [FKP18].

The existence of such an algorithm is surprising but in hindsight not entirely unexpected: remember that, in formal language theory, when we are given a word and a nondeterministic finite automaton, then we can evaluate the automaton on the word with tractable combined complexity by determinizing the automaton "on the fly", i.e., computing at each position of the word the set of states where the automaton can be. Our algorithm generalizes this intuition, and extends it to the task of enumerating mappings without duplicates: our overall approach is to construct a kind of product of the input document with the extended VA, similarly to [FRU$^+$18]. We then use several tricks to ensure the constant delay bound despite nondeterminism; in particular we precompute a

jump function that allows us to skip quickly the parts of the document where no variable can be assigned. To avoid determinizing the input VA, our idea for this is to efficiently enumerate at each position the possible sets of markers that can be assigned by the VA: we do so by enumerating paths in the VA, relying on the fact that the VA is sequential so these paths are acyclic. The challenge is that the same set of markers can be captured by many different paths, but we explain how we can explore efficiently the set of distinct paths with a technique known as *flashlight search* [MS16, RT75]: the key idea is that we can efficiently determine which partial sets of markers can be extended to the label of a path. The resulting algorithm is rather simple and has no large hidden constants.

Note that our enumeration algorithm does not contradict the counting hardness results of [FRU+18]: while our algorithm *enumerates* mappings with constant delay and without duplicates, we do not see a way to adapt it to *count* the mappings efficiently. This is similar to the enumeration and counting problems for maximal cliques: one can enumerate maximal cliques with polynomial delay [TIAS77], but counting them is #P-hard [Val79b].

One last contribution of [20] is to present a prototype implementation of the enumeration algorithm sketched above which is available online as open-source software[2]. We evaluate this software experimentally for different types of queries. The results show that our approach can be implemented in practice and run efficiently. Our prototype is used in a project called "Spanner workbench" currently under development at the *Technion Data and Knowledge (TD&K) Laboratory* led by Prof. Benny Kimelfeld whose aim it is to implement a public toolbox for document spanners. Our implementation is part of the standard library of extraction functions provided by that project.

[2]https://github.com/PoDMR/enum-spanner-rs

4. Conclusion

This chapter concludes the first part of this thesis which gave a survey of most of my work over the last years.

In the remainder of this chapter, I will quickly survey some of the work that I have done that does not fit the two main lines of counting and compilation and which have thus been left out before. I will then, in Section 4.2, also give an outlook presenting some specific questions that I think would be good continuations of the work presented in this thesis.

4.1. Other Work

Since I restricted myself to counting complexity and knowledge compilation in this thesis, I have left out some of my work that does not fit into these two areas. So let me give a very quick mention of this work here; the interested reader will find the details in the respective papers.

I have written a sequence of papers with Mike Behrisch, Miki Hermann and Gernot Salzer during my postdoc at École Polytechnique [32, 31, 30] which was later summarized in one journal paper [4]. In those papers, we classify certain types of minimization problems over conjunctions of Boolean constraints with respect to their approximation properties. This can be seen as an generalization of by now classical work of [KST97], see also [CKS01].

There is also joint work with Daniel Le Berre, Pierre Marquis and my then student Romain Wallon in which we study a peculiar phenomenon of solvers for pseudo-Boolean (PB) constraints [16]. These solvers are modeled after more established conflict-driven clause learning solvers for CNF-formulas but use the stronger proof system of cutting planes instead of resolution. It turns out that implementations of this lead to the creation of irrelevant literals in the learned constraints, i.e., literals whose assigned values (whatever they are) never change the truth value of the constraint. We show that those irrelevant literals exist and that in certain cases they slow down solvers dramatically. In his PhD thesis [Wal20], Romain extended this line of work by proposing adapted procedures in conflict analysis that allow avoiding irrelevant literals to a certain extent.

Together with Sebastian Skritek, we analyzed so-called well-designed pattern trees, a formalization of an important well-behaved fragment of SPARQL, the query language for RDF graphs, which is a data format that appears in the context of the semantic web. The semantics of well-designed pattern trees is specifically designed to allow accessing incomplete data sources which makes their evaluation hard and complicates the understanding of their complexity. In [21], we study the complexity of evaluating such well-designed pattern trees. A short time before our paper, a characterization of

the classes of well-designed pattern trees that can be evaluated in polynomial time was given [Rom18]. However, projection—a central feature of many query languages—was not considered in that work. We work towards closing this gap by giving a characterization of all tractable classes of so-called simple well-designed pattern trees with projection (under some common complexity theoretic assumptions). Since well-designed pattern trees correspond to the fragment of well-designed {AND, OPTIONAL}-SPARQL queries, this gives a complete description of the tractable classes of queries with projections in this fragment that can be characterized by the underlying graph structures.

While at first sight, this work has no direct connection to my other papers, there is indeed an intimate technical connection to my work on counting complexity, in particular [35]. Indeed, many of the combinatorial objects and techniques that turn out important to understand well-designed pattern trees are closely related to those I had used in that work but had to be tweaked to fit the different setting.

4.2. Outlook

There are several topics that I think would be interesting continuations of the work presented in this thesis; I will sketch some of them in this section. Generally, I am planning to continue moving my research somewhat closer to practical applications. The idea is not to completely change my research to become a solver writer or similar—this would not correspond to my strengths or background—but to work on some more theoretical questions that are directly relevant to my more practical colleagues. Note that some of my recent work goes in this direction, e.g. our implementation of the algorithm in [20] is used by the group of Benny Kimelfeld, and the paper [14] gives a theoretical analysis of an implemented system. Below, I give a mix of such more applied and purely theoretical questions.

Generally, I believe that there are still more areas in which a knowledge compilation perspective is useful. Therefore, I will continue looking out for such opportunities in the future. However, since this is of course inherently not predictable, let me present some more concrete plans for the future here.

Determinism. One important task in core knowledge compilation is certainly better understanding determinism in DNNF, both for algorithmic upper bounds and lower bounds. On the one hand, there are no algorithms that allow to use the full power of deterministic DNNF; in particular, all implementations compile to restricted fragments like so-called decision-DNNF or SDD [Dar11]. It would be interesting to see if there are good algorithms to compile into the larger fragment of d-DNNF without such restrictions. On the other hand, it would be good to improve our theoretical understanding of ways to show lower bounds for d-DNNF. Note that the only lower bound separating general DNNF from d-DNNF is by a function introduced by Sauerhoff [Sau03] and it is unclear how to generalize Sauerhoff's technique to other, more interesting functions. In particular, it would be intriguing to show that there are monotone DNF that do not have small d-DNNF in the hope that this might lead to lower bounds for so-called lineages in

database theory, see e.g. [JS13, Mon20] for background in this direction.

Note that there is also a relation between the power of determinism and the question if structured d-DNNF can be complemented efficiently: in [BF21] it is shown that a function f can represented by a small SDD if and only if f and $\neg f$ can both be represented by small d-DNNF respecting the same v-tree. So separating SDD and structured d-DNNF requires in a sense showing that there is a function f such that f is represented by a small structured d-DNNF while $\neg f$ does not have a small d-DNNF representation respecting the same v-tree.

Finally, one question around determinism is if there are good algorithms that compile into DNNF that are not deterministic. Since, as discussed in Section 2.4.1, determinism generally forces DNNF to be exponentially more verbose, for some applications that do not require efficient counting it would be good to be able to compile into nondeterministic DNNF to hopefully get smaller representations. However, besides [OD17] there seems to be no work on that direction whatsoever. The main problem is that it would require to leave the well-understood paradigms of DPLL-style compilation and also the known bottom-up approaches which all compile to deterministic DNNF. Thus, it is at the same time challenging but also highly interesting to understand how in theory but also practical applications nondeterminism can be used in DNNF compilation.

Approximation. There are several questions regarding approximate knowledge compilation that should be explored. First, as discussed in Section 2.5, one can show good lower bounds for d-DNNF that approximate functions. It would be good to develop techniques that allow showing lower bounds for general nondeterministic DNNF. Note that the technique from [15] inherently does not work in this setting because we crucially use the fact that a d-DNNF leads to a version of Equation 2.1 in which the rectangles share no models. As a consequence, when each individual rectangle contains false positives, then we know that the false positives must be disjoint for the different rectangles and thus there must be many false positives in total. This then leads to the desired lower bound. For general DNNF however, the false positives in all rectangles could be the same, which makes the argument break down. It is an intriguing question to see how this problem can be avoided.

Another question is finding problems for which approximation *can* improve the efficiency of knowledge compilation. Besides the algorithm from [CT20] on bounded pathwidth Bayes classifiers, there seem to be no positive results in this direction. Finding more use cases where approximation helps to compile efficiently and in particular also giving efficient algorithms for this, could be very interesting.

Related to the lower bound results on arithmetic circuits in Section 3.4, it is very natural to apply approximation there. For example, instead of compiling a classifier for a Bayesian network, it might be useful to approximately compile the probability distribution of the network itself. Note that in this setting, it is not even obvious which distance measure between the original distribution and the approximating circuit to use. So one would first have to define this quality criterion to make the approximation useful and then see in which settings one can approximate efficiently.

Fine-grained complexity. The lower bound techniques of Section 2.4 are inherently such that they lose a polynomial factor with respect to the real lower bound. Intuitively, this is because in that setting we are only counting the rectangles computed in the circuit which only gives a bound on the number of gates in a "middle" layer of the circuit. However, each rectangle has to also be computed in the circuit which requires additional gates. This leads to the observation that the techniques of Section 2.4 can inherently not show tight bounds for functions that have DNNF representations of polynomial size. In this setting all lower bounds will be off by a polynomial factor in the number of variables. For example, with this approach, it appears impossible to separate functions having representations with linear size DNNF from such where all DNNF representations are of superlinear size. However, such bounds would be desirable, in particular from the perspective of database theory, an area in which often linear instead of mere polynomial time complexity is seen as a yardstick for efficient computation. So can the lower bound techniques be refined?

Very recently, I have used classical techniques on partial derivatives from arithmetic circuit complexity to show tight cubic bounds for a function. The underlying idea is that this technique allows to reduce lower bound questions on a single function to lower bounds for several functions that have to be computed in the same circuit. This makes tight lower bounds potentially easier. However, the approach so far is quite specialized for the function I considered, and it remains to be seen how it can be generalized.

I am also currently working on fine-grained questions in query answering in database theory using techniques from so-called fine-grained complexity theory. This area mostly deals with lower bounds for problems which are known to be solvable in polynomial time and tries to determine the exact exponent of optimal algorithms, see e.g. [Wil18] for a recent survey on this currently very dynamic field. In database theory, fine-grained complexity has been taken up over the last few years in order to show for which problems one can hope to improve on current algorithms, see e.g. the survey [BGS20]. I expect that fine-grained complexity will have a great and lasting impact in the area. With Nofar Carmeli, I am currently working on ordered enumeration for query answers where we make connections to fine-grained complexity and I have some other ideas in this direction.

Proof complexity. In the medium turn, I am also planning to extend my research more in the direction of proof complexity. I think that the lower bound questions studied there are sufficiently close to similar questions in knowledge compilation such that I think I can make contributions in that field. Here are two concrete questions I am planning to work on: first, I would like to understand how general the link between knowledge compilation and proof complexity is on a technical level. Remember that in [13], building on [IRSS19], we made an explicit connection between DNNF size for satisfiable Tseitin-formulas and regular resolution refutations of unsatisfiable Tseitin-formulas. It is an intriguing question if there are similar links for other classes of CNF-formulas. Can this lead to a better understanding of both regular resolution and compilation bounds? Note that in a similar direction, the PERM_nformulas studied in [18] are in a sense maximal satisfiable versions of pigeonhole formulas which are classical hard instances in proof complexity. For these formulas lower bounds have long been known to be hard for resolution, see e.g. [BN21].

Are there deeper connections between these two lower bounds? Can one of them be used to prove the other?

Another question in proof complexity that I would like to better understand is the complexity of FBDD decision lists which are a generalization of a technical tool in [14]. Decision lists are representations of Boolean functions that are evaluated by working along a list until a certain evaluation criterion is met and the value can be read off directly. Such decision lists have long been studied in pure theoretical computer science and have over the last years become important in the study of QBF proof complexity. This is because lower bounds on them can be lifted to lower bounds in different QBF proof systems. In this direction, a better understanding of FBDD decision lists would lead to lower bounds for a stronger QBF proof system than that in [14] which corresponds to solvers that change the variable order of OBDDs while refuting an input. Beyond this, showing lower bounds for such decision lists is also a compelling theoretical question from a communication complexity perspective.

Part II.

Some Technical Details

In this part of this thesis, we will see some of the technical details of the papers described in Part I. The aim is not to be exhaustive—this would take far too much space—but to sample some of the techniques I have used and some of the areas I have worked in over the years. The chapters in this part are essentially the papers in which the results appeared, sometimes with some details changed. For example, proofs that were delegated to an appendix in the publications were often integrated in the main text, or there are slight typographic changes or similar.

The results that are presented here were chosen using different criteria. A first aim was to present a certain breadth of topics and techniques. Instead of presenting several similar papers in the same direction, there is mostly just one result per direction. I have also prioritized work that was more "important"[1] in the sense that it presented insights or techniques that later were of use for other work by me or others. Finally, there is also the criterion of which papers I personally like the most and which techniques I find the most interesting. Note that this leads to a slight preference for newer papers; I always tend to be most excited about my newest work.

Concretely, I will present work from the following papers:

- In Chapter 5, we will see the proof of the dichotomy result of [28] that was discussed in Chapter 1.2. In contrast to most of my other work, the underlying techniques in that work are *not* combinatorial, in particular there is essentially no graph theory involved. Instead, the crucial arguments are algebraic and argue by properties of polynomials, which I think makes the paper quite interesting to read. At least I enjoyed revisiting it while preparing this thesis.

- Chapter 6 gives the technical details of [27] on the connection between DNNF lower bounds and communication complexity. As we have seen in Part I, this result is a crucial building block for much of my work afterwards, which is why I chose to present it here. As the reader will see, the technical argument is actually quite simple, so I tend to think that the main contribution of [27] is not technical but conceptual. Making the connection to communication complexity explicit allowed solving some open questions rather easily by using results from the literature.

- In Chapter 7, we will see the application of [27] to approximate knowledge compilation, showing the results from [15] discussed in Section 2.5. There we will see that even though the lower bound framework by communication complexity is rather simple, actually using it sometimes requires quite nontrivial—and in this case quite pretty—combinatorial arguments.

- The application of knowledge compilation in proof complexity from [13] that was sketched in Section 3.2 will be presented in Section 8. The main technical contribution in that paper is a fine understanding of edge sets whose deletion leaves a graph connected and how their size is connected to treewidth. The proofs in that part felt to me like a return to my roots in structural graph theory during my undergraduate degree.

[1] As a theorist, I am using this term very loosely.

- Finally, in Chapter 9, we will see in more detail the enumeration algorithm for document spanners presented in Section 3.5.3. The connection to knowledge compilation is a little more hidden in this, but there is an important part played by what we call *mapping DAGs* there that are essentially a form of nondeterministic multi-valued decision diagrams.

All these chapters are largely independent from each other—in particular, they all contain all necessary preliminaries, even though this means that several concepts are introduced in more than one chapter—, so the reader can choose freely which subjects interest them most and should not feel obliged to read them all. That said, the different chapters present different facets of my work, so to get a feeling for what I am doing in my research, it might be useful to skim several of them.

5. Counting Answers to Existential Positive Queries: A Complexity Classification

In this chapter, we will see the details of the trichotomy result for counting satisfying assignments of existential positive queries which was introduced in Section 1.2 and which originally appeared in [28].

5.1. Preliminaries

5.1.1. Basic definitions and notions

Note that $\cdot$ is sometimes used for multiplication of real numbers.

Polynomials. We remind the reader of some basic facts about polynomials which we will use throughout the paper. Here, a univariate polynomial p in a variable x is a function $p(x) = \sum_{i=0}^{d} a_i x^i$ where $d \geq 0$, each $a_i \in \mathbb{R}$ and $a_d \neq 0$, or the *zero polynomial* $p(x) = 0$. The a_i are called *coefficients* of p. The degree of a polynomial is defined as $-\infty$ in the case of the zero polynomial, and as d otherwise. Let $(x_0, y_0), \ldots, (x_n, y_n)$ be $n+1$ pairs of real numbers. Then there is a uniquely determined polynomial of degree at most n such that $p(x_i) = y_i$ for each i; consequently, a polynomial p of degree n that has at least $n+1$ zeroes (where a *zero* is a value x such that $p(x) = 0$) is the zero polynomial. If all x_i and y_i are rational numbers, then the coefficients a_i of this polynomial are rational numbers as well; moreover, the a_i can be computed in polynomial time.

Logic. We assume basic familiarity with the syntax and semantics of first-order logic. In this article, we focus on relational first-order logic where equality is not built-in to the logic. Hence, each *vocabulary/signature* under discussion consists only of relation symbols. We assume structures under discussion to be *finite* (that is, have finite universe); nonetheless, we sometimes describe structures as *finite* for emphasis. We assume that the relations of structures are represented as lists of tuples. We use the letters $\mathbf{A}$, $\mathbf{B}$, ... to denote structures, and the corresponding letters A, B, ... to denote their respective universes. When τ is a signature, we use $\mathbf{I}_\tau$ to denote the τ-structure with universe $\{a\}$ and where each relation symbol $R \in \tau$ has $R^{\mathbf{I}} = \{(a, \ldots, a)\}$. When $\mathbf{A}, \mathbf{B}$ are structures over the same signature τ, a *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ is a mapping $h : A \to B$ such that, for each $R \in \tau$ and each tuple $(a_1, \ldots, a_k) \in R^{\mathbf{A}}$, it holds that $(h(a_1), \ldots, h(a_k)) \in R^{\mathbf{B}}$.

We use the term *fo-formula* to refer to a first-order formula. An *ep-formula* (short for *existential positive formula*) is a fo-formula built from *atoms* (by which we refer to predicate applications of the form $R(v_1, \ldots, v_k)$, where R is a relation symbol and the

v_i are variables), conjunction ($\wedge$), disjunction ($\vee$), and existential quantification ($\exists$). A *pp-formula* (short for *primitive positive formula*) is defined as an ep-formula where disjunction does not occur. An fo-formula is *prenex* if it has the form $Q_1 v_1 \dots Q_n v_n \theta$ where θ is quantifier-free, that is, if all quantifiers occur in the front of the formula. The set of free variables of a formula ϕ is denoted by $\text{free}(\phi)$ and is defined as usual; a formula ϕ is a *sentence* if $\text{free}(\phi) = \emptyset$.

We now present some definitions and conventions that are not totally standard. A primary concern in this article is in counting satisfying assignments of fo-formulas on a finite structure. The count is sensitive to the set of variables over which assignments are considered; and, we will sometimes (but not always) want to count relative to a set of variables that is strictly larger than the set of free variables. Hence, we will often associate with each fo-formula ϕ a set V of variables called the *liberal variables*, denoted by $\text{lib}(\phi)$, which is required to be a superset of $\text{free}(\phi)$, that is, we require $\text{lib}(\phi) \supseteq \text{free}(\phi)$. Note that $\text{lib}(\phi)$ may contain variables that do not occur at all in atoms of ϕ. To indicate that V is the set of liberal variables of ϕ, we often use the notation $\phi(V)$; we also use $\phi(v_1, \dots, v_n)$, where the v_i are a listing of the elements of V. Relative to a formula $\phi(V)$, when $\mathbf{B}$ is a structure, we will use $\phi(\mathbf{B})$ to denote the set of assignments $f : V \to B$ such that $\mathbf{B}, f \models \phi$. *We assume that, in each prenex formula with liberal variables associated with it, no variable is both liberal and quantified.* We call an fo-formula ϕ *free* if $\text{free}(\phi) \neq \emptyset$, and *liberal* if $\text{lib}(\phi)$ is defined and $\text{lib}(\phi) \neq \emptyset$.

Example 5.1. *Let us consider the formula* $\phi(x, y, z) = R(x, y) \vee S(y, z)$. *As indicated above, the notation* $\phi(x, y, z)$ *is used to indicate that* $\text{lib}(\phi) = \{x, y, z\}$. *As* $\text{free}(\phi) = \{x, y, z\}$, *we have* $\text{lib}(\phi) = \text{free}(\phi)$. *Define* $\psi(x, y, z) = R(x, y)$ *and* $\psi'(x, y, z) = S(y, z)$. *By the notation* $\psi(x, y, z)$, *we indicate that* $\text{lib}(\psi) = \{x, y, z\}$; *likewise, it holds that* $\text{lib}(\psi') = \{x, y, z\}$. *Notice that* $\text{free}(\psi) = \{x, y\}$, *so we have that* $\text{lib}(\psi)$ *is a proper superset of* $\text{free}(\psi)$; *in fact, the variable* $z \in \text{lib}(\psi)$ *does not occur at all in an atom of* ψ. *Define also* $\theta(x, y) = R(x, y)$; *by the notation* $\theta(x, y)$, *we indicate that* $\text{lib}(\theta) = \{x, y\}$.

Observe that, for any structure $\mathbf{B}$, *we have* $\phi(\mathbf{B}) = \psi(\mathbf{B}) \cup \psi'(\mathbf{B})$ *(and hence* $|\phi(\mathbf{B})| = |\psi(\mathbf{B}) \cup \psi'(\mathbf{B})|$*). Observe, however, that for any structure* $\mathbf{B}$ *where* $\theta(\mathbf{B})$ *is non-empty, it does not hold that* $\phi(\mathbf{B}) = \theta(\mathbf{B}) \cup \psi'(\mathbf{B})$, *since* $\phi(\mathbf{B})$ *contains only assignments defined on* $\text{lib}(\phi) = \{x, y, z\}$, *whereas* $\theta(\mathbf{B})$ *contains only assignments defined on* $\text{lib}(\theta) = \{x, y\}$.

pp-formulas. It is well-known [CM77] that there is a correspondence between prenex pp-formulas and relational structures. In particular, each prenex pp-formula $\phi(S)$ (on signature τ) with $\text{lib}(\phi) = S$ may be viewed as a pair $(\mathbf{A}, S)$ consisting of a structure $\mathbf{A}$ (on τ) and a set S; the universe A of $\mathbf{A}$ is the union of S with the variables appearing in ϕ, and the following condition defines the relations of $\mathbf{A}$: for each $R \in \tau$, a tuple $(a_1, \dots, a_k) \in A^k$ is in $R^{\mathbf{A}}$ if and only if $R(a_1, \dots, a_k)$ appears in ϕ. In the other direction, such a pair $(\mathbf{A}, S)$ can be viewed as a prenex pp-formula $\phi(S)$ where all variables in $A \setminus S$ are quantified and the atoms of ϕ are defined according to the above condition. A basic known fact [CM77] that we will use is that when $\phi(S)$ is a pp-formula corresponding to the pair $(\mathbf{A}, S)$, $\mathbf{B}$ is an arbitrary structure, and $f : S \to B$ is an arbitrary map, it holds that $\mathbf{B}, f \models \phi(S)$ if and only if there is an extension f' of f that is a homomorphism

from $\mathbf{A}$ to $\mathbf{B}$. *We will freely interchange between the structure view and the usual notion of a prenex pp-formula.* For a pp-formula specified as a pair $(\mathbf{A}, S)$, we typically assume that $S \subseteq A$.

Example 5.2. *Consider the pp-formula* $\phi(x, x', y, z) = \exists y' \exists u \exists v \exists w (E(x, x') \wedge E(y, y') \wedge F(u, v) \wedge G(u, w))$. *The notation* $\phi(x, x', y, z)$ *indicates that* $\mathrm{lib}(\phi) = \{x, x', y, z\}$. *Note that* $\mathrm{free}(\phi) = \{x, x', y\}$. *To convert* ϕ *to a structure* $\mathbf{A}$*, we take the universe* A *of* $\mathbf{A}$ *to be the union of* $\mathrm{lib}(\phi)$ *with all variables appearing in* ϕ*, so* $A = \{x, x', y, z, y', u, v, w\}$. *Then, we define the relations as just described above, so* $E^{\mathbf{A}} = \{(x, x'), (y, y')\}$, $F^{\mathbf{A}} = \{(u, v)\}$, *and* $G^{\mathbf{A}} = \{(u, w)\}$. *The resulting pair representation of* ϕ *is* $(\mathbf{A}, \{x, x', y, z\})$.

Two structures are *homomorphically equivalent* if each has a homomorphism to the other. A structure is a *core* if it is not homomorphically equivalent to a proper substructure of itself. A structure $\mathbf{B}$ is a *core* of a structure $\mathbf{A}$ if $\mathbf{B}$ is a substructure of $\mathbf{A}$ that is a core and is homomorphically equivalent to $\mathbf{A}$. It is known that all cores of a structure are isomorphic and hence one sometimes speaks of *the* core of a structure.

For a prenex pp-formula $(\mathbf{A}, S)$ on signature τ, we define its *augmented structure*, denoted by $\mathsf{aug}(\mathbf{A}, S)$, to be the structure over the expanded vocabulary $\tau \cup \{R_a \mid a \in S\}$ (understood to be a disjoint union) where $R_a^{\mathsf{aug}(\mathbf{A},S)} = \{a\}$; we define the *core* of the pp-formula $(\mathbf{A}, S)$ to be the core of $\mathsf{aug}(\mathbf{A}, S)$.

The following fundamental facts on pp-formulas will be used throughout.

Theorem 5.3 (follows from [CM77])**.** *Suppose that each of the pairs* $(\mathbf{A}, V)$, $(\mathbf{B}, V)$ *is a prenex pp-formula. The formula* $(\mathbf{B}, V)$ *logically entails the formula* $(\mathbf{A}, V)$ *if and only if there exists a homomorphism from the structure* $\mathsf{aug}(\mathbf{A}, V)$ *to the structure* $\mathsf{aug}(\mathbf{B}, V)$. *The formulas* $(\mathbf{A}, V)$, $(\mathbf{B}, V)$ *are logically equivalent if and only if they have isomorphic cores, or equivalently, when* $\mathsf{aug}(\mathbf{A}, V)$ *and* $\mathsf{aug}(\mathbf{B}, V)$ *are homomorphically equivalent.*

ep-formulas. In order to discuss ep-formulas, we will employ the following terminology. An ep-formula is *disjunctive* if it is the disjunction of prenex pp-formulas; when ϕ is a disjunctive ep-formula with $\mathrm{lib}(\phi)$ defined, we typically assume that each of the pp-formulas ψ that appear as disjuncts of ϕ has $\mathrm{lib}(\psi)$ defined as $\mathrm{lib}(\phi)$. (In this way, for an arbitrary finite structure $\mathbf{B}$, it holds that $|\phi(\mathbf{B})| = |\bigcup_\psi \psi(\mathbf{B})|$, where the union is over all such disjuncts ψ.) An ep-formula is *all-free* if it is disjunctive and each pp-formula appearing as a disjunct is free. An ep-formula $\phi(S)$ is *normalized* if it is disjunctive and for each sentence disjunct $(\mathbf{A}, S)$ and any other disjunct $(\mathbf{A}', S)$, there is no homomorphism from $\mathsf{aug}(\mathbf{A}, S)$ to $\mathsf{aug}(\mathbf{A}', S)$ (equivalently, there is no homomorphism from $\mathbf{A}$ to $\mathbf{A}'$). It is straightforward to verify that there is an algorithm that, given an ep-formula, outputs a logically equivalent normalized ep-formula.

Graphs. To every prenex pp-formula $(\mathbf{A}, S)$ we assign a graph whose vertex set is $A \cup S$ and where two vertices are connected by an edge if they appear together in a tuple of a relation of $\mathbf{A}$. A prenex pp-formula $(\mathbf{A}, S)$ is called *connected* if its graph is connected. A prenex pp-formula $(\mathbf{A}', S')$ is a *component* of a prenex pp-formula $(\mathbf{A}, S)$ over the same

signature τ if there exists a set C that forms a connected component of the graph of $(\mathbf{A}, S)$, where:

- $S' = S \cap C$.
- For each relation $R \in \tau$, a tuple $(a_1, \ldots, a_k)$ is in $R^{\mathbf{A}'}$ if and only if $(a_1, \ldots, a_k) \in R^{\mathbf{A}} \cap C^k$.

Note that when this holds, the graph of $(\mathbf{A}', S')$ is the connected component of the graph of $(\mathbf{A}, S)$ on vertices C. We will use the fact that, if $\phi(V)$ is a prenex pp-formula and $\phi_1(V_1), \ldots, \phi_c(V_c)$ is a list of its components, then for any finite structure $\mathbf{B}$, it holds that $|\phi(\mathbf{B})| = \prod_{i=1}^{c} |\phi_i(\mathbf{B})|$.

Example 5.4. *Let ϕ be the free prenex pp-formula from Example 5.2, and let $(\mathbf{A}, S)$ be the pair representation given there. The connected components of the graph of $(\mathbf{A}, S)$ are $\{x, x'\}$, $\{y, y'\}$, $\{z\}$, and $\{u, v, w\}$. There are thus four components of the formula $(\mathbf{A}, S)$; they are $(\mathbf{A}'_{\{x,x'\}}, \{x, x'\})$, $(\mathbf{A}'_{\{y,y'\}}, \{y\})$, $(\mathbf{A}'_{\{z\}}, \{z\})$, and $(\mathbf{A}'_{\{u,v,w\}}, \emptyset)$ (respectively), where each $\mathbf{A}'_C$ is the structure $\mathbf{A}'$ defined above, with respect to the set C.*

Written logically, these four components are $\psi_1(x, x') = E(x, x')$, $\psi_2(y) = \exists y' E(y, y')$, $\psi_3(z) = \top$, and $\psi_4(\emptyset) = \exists u \exists v \exists w (F(u, v) \wedge G(u, w))$, respectively. Here, $\top$ denotes the empty conjunction (considered to be true).

5.1.2. Counting complexity

Throughout, we use Σ to denote an alphabet over which strings are formed. All problems to be considered are viewed as counting problems. So, a *problem* is a mapping $Q : \Sigma^* \to \mathbb{N}$. We view decision problems as problems where, for each $x \in \Sigma^*$, it holds that $Q(x)$ is equal to 0 or 1. A *parameterization* is a mapping $\kappa : \Sigma^* \to \Sigma^*$. A parameterized problem is a pair (Q, κ) consisting of a problem Q and a parameterization κ. Throughout, by π_i we denote the operator that projects a tuple onto its ith coordinate.

A partial function $T : \Sigma^* \to \mathbb{N}$ is *polynomial-multiplied* with respect to a parameterization κ if there exists a computable function $f : \Sigma^* \to \mathbb{N}$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that, for each $x \in \mathrm{dom}(T)$, it holds that $T(x) \leq f(\kappa(x))p(|x|)$.

We now give a definition of FPT-computability for partial mappings.

Definition 5.5. Let $\kappa : \Sigma^* \to \Sigma^*$ be a parameterization. A partial mapping $r : \Sigma^* \to \Sigma^*$ is *FPT-computable* with respect to κ if there exist a polynomial-multiplied function $T : \Sigma^* \to \mathbb{N}$ (with respect to κ) with $\mathrm{dom}(T) = \mathrm{dom}(r)$ and an algorithm A such that, for each string $x \in \mathrm{dom}(r)$, the algorithm A computes $r(x)$ within time $T(x)$; when this holds, we also say that r is *FPT-computable* with respect to κ *via* A.

As is standard, we may and do freely interchange among elements of Σ^*, $\Sigma^* \times \Sigma^*$, and $\mathbb{N}$. We define FPT to be the class that contains a parameterized problem (Q, κ) if and only if Q is FPT-computable with respect to κ.

We now introduce a notion of reduction for counting problems, which is a form of Turing reduction. We use $\wp_{\mathsf{fin}}(A)$ to denote the set containing all finite subsets of A.

Definition 5.6. A *counting FPT-reduction* from a parameterized problem (Q, κ) to another (Q', κ') consists of a computable function $h : \Sigma^* \to \wp_{\mathsf{fin}}(\Sigma^*)$, and an algorithm A such that:

- on an input x, A may make oracle queries of the form $Q'(y)$ with $\kappa'(y) \in h(\kappa(x))$, and
- Q is FPT-computable with respect to κ via A.

We use clique to denote the decision problem where (k, G) is a yes-instance when G is a graph that contains a clique of size $k \in \mathbb{N}$. By #clique we denote the problem of counting, given (k, G), the number of k-cliques in the graph G. The parameterized versions of these problems, denoted by p-clique and p-#clique, are defined via the parameterization $\pi_1(k, G) = k$.

5.1.3. Counting case complexity

We employ the framework of *case complexity* to develop some of our complexity results. We present the needed elements of this framework for counting problems. The definitions and results here are due to [35], are based on the theory of [Che14b], and are presented here for the sake of self-containment; see those articles for further discussion and motivation of the framework.

The case complexity framework was developed to prove results on restricted versions of parameterized problems where not all values of the parameter are permitted. This type of restricted problem arises naturally in query answering problems, where one often restricts the queries that are admissible, as is done here (for other examples, see [DJ04b, Gro07, Che14b]).

The case complexity framework provides a notion of *case problem* and a notion of reduction between case problem. A case problem was originally [Che14b] defined as a language Q of pairs (that is, a subset of $\Sigma^* \times \Sigma^*$) where the first element of each pair is ultimately viewed as the parameter, along with a set $S \subseteq \Sigma^*$ restricting the permitted parameter values. In this article, as we are dealing with counting complexity, in lieu of considering languages, we will consider mappings $\Sigma^* \times \Sigma^* \to \mathbb{N}$. (Of course, a language of pairs can be naturally viewed as such a mapping by taking its characteristic function.)

One benefit of the framework is that the notion of reduction does not rely on any form of computability assumption on the sets S involved. Thus, in comparing case problems using this notion of reduction, one does not need to discuss the computability status of these sets S, even though in general, it is usual that authors ultimately assume some form of computability on these sets (typically computable enumerability or computability).

Let us turn to the formal presentation of the framework. A *case problem* consists of a problem $Q : \Sigma^* \times \Sigma^* \to \mathbb{N}$ and a subset $S \subseteq \Sigma^*$, and is denoted $Q[S]$. Note that, although a problem above is defined as a mapping from Σ^* to $\mathbb{N}$, here we work with a problem that is a mapping from $\Sigma^* \times \Sigma^*$ to $\mathbb{N}$; this is natural in the current paper, where an input to the studied problem consists of two parts, a formula and a structure. Note that a mapping $\Sigma^* \times \Sigma^* \to \mathbb{N}$ can be naturally viewed as a mapping $\Sigma^* \to \mathbb{N}$, as there

are natural and well-known ways to encode the elements of $\Sigma^* \times \Sigma^*$ as elements of Σ^*. For each case problem $Q[S]$, we define $\mathsf{param}\text{-}Q[S]$ as the parameterized problem (P, π_1) where $P(s, x)$ is defined as equal to $Q(s, x)$ if $s \in S$, and as 0 otherwise.

We have the following reduction notion for case problems.

Definition 5.7. A *counting slice reduction* from a case problem $Q[S]$ to a second case problem $Q'[S']$ consists of

- a computably enumerable language $U \subseteq \Sigma^* \times \wp_{\mathsf{fin}}(\Sigma^*)$, and
- a partial function $r : \Sigma^* \times \wp_{\mathsf{fin}}(\Sigma^*) \times \Sigma^* \to \Sigma^*$ that has domain $U \times \Sigma^*$ and is FPT-computable with respect to (π_1, π_2) via an algorithm A that, on input (s, T, y), may make queries of the form $Q'(t, z)$ where $t \in T$,

such that the following conditions hold:

- (coverage) for each $s \in S$, there exists $T \subseteq S'$ such that $(s, T) \in U$, and
- (correctness) for each $(s, T) \in U$, it holds (for each $y \in \Sigma^*$) that $Q(s, y) = r(s, T, y)$.

Let us provide some intuition for this definition. Here, when discussing an instance (s, y) of a case problem, we refer to the first part s as the parameter. The role of U is to provide all pairs (s, T) such that instances (of the first problem) with parameter s can be reduced to instances (of the second problem) whose parameters lie in T. Correspondingly, the *coverage* condition posits that each $s \in S$ is covered by the second set S' in the sense that there exists a pair $(s, T) \in U$ with $T \subseteq S'$. The partial function r is the actual reduction; given a pair $(s, T) \in U$ along with a string y, it computes the value $Q(s, y)$—this is what the *correctness* condition asserts. As here in this article we are dealing with counting complexity, we permit a form of Turing reduction; so, the algorithm A of the partial function r, upon being given a triple (s, T, y), may make (possibly multiple) queries to the second problem, so long as the queries are about instances whose parameter falls into T.

We have the following key property of counting slice reducibility.

Theorem 5.8. *[35] Counting slice reducibility is transitive.*

The following theorem shows that, from a counting slice reduction, one can obtain complexity results for the corresponding parameterized problems.

Theorem 5.9. *[35] Let $Q[S]$ and $Q'[S']$ be case problems. Suppose that $Q[S]$ counting slice reduces to $Q'[S']$, and that both S and S' are computable. Then* $\mathsf{param}\text{-}Q[S]$ *counting FPT-reduces to* $\mathsf{param}\text{-}Q'[S']$.

5.1.4. Classification of pp-formulas

We present the complexity classification of pp-formulas previously presented in [35]. The following definitions are adapted from that article. Let $(\mathbf{A}, S)$ be a prenex pp-formula, let $\mathbf{D}$ be the core thereof, and let $G = (D, E)$ be the graph of $\mathbf{D}$. An $\exists$-*component* of

$(\mathbf{A}, S)$ is a graph of the form $G[V']$ where there exists $V \subseteq D$ that is the vertex set of a component of $G[D \setminus S]$ and V' is the union of V with all vertices in S having an edge to V. Define $\mathsf{contract}(\mathbf{A}, S)$ to be the graph on vertex set S obtained by starting from $G[S]$ and adding an edge between any two vertices that appear together in an $\exists$-component of $(\mathbf{A}, S)$.

Let Φ be a set of prenex pp-formulas. Let us say that Φ satisfies the *contraction condition* if the graphs in the set $\mathsf{contract}(\Phi) := \{\mathsf{contract}(\phi) \mid \phi \in \Phi\}$ are of bounded treewidth. Let us say that Φ satisfies the *tractability condition* if it satisfies the contraction condition and, in addition, the cores of Φ are of bounded treewidth; here, the treewidth of a prenex pp-formula is defined as that of its graph. We omit the definition of treewidth, as it is both well-known and not needed to understand the main technical proof of this article (which is in Section 5.4).

Definition 5.10. We define count to be the problem that maps a pair $(\phi(V), \mathbf{B})$ consisting of a fo-formula and a finite structure to the value $|\phi(\mathbf{B})|$.

Theorem 5.11. *[35] Let Φ be a set of prenex pp-formulas that satisfies the tractability condition. Then, the restriction of $\mathsf{param\text{-}count}[\Phi]$ to $\Phi \times \Sigma^*$ is an FPT-computable partial function.*

Theorem 5.12. *[35] Let Φ be a set of prenex pp-formulas of bounded arity that does not satisfy the tractability condition.*

1. *If Φ satisfies the contraction condition, then it holds that $\mathsf{count}[\Phi]$ and $\mathsf{clique}[\mathbb{N}]$ are interreducible, under counting slice reductions.*
2. *Otherwise, there exists a counting slice reduction from $\#\mathsf{clique}[\mathbb{N}]$ to $\mathsf{count}[\Phi]$.*

We say that a set of formulas Φ has *bounded arity* if there exists a constant $k \geq 1$ that upper bounds the arity of each relation symbol appearing in a formula in Φ.

5.2. Main theorems

The following theorem, which we call the *equivalence theorem* and which is proved in Section 5.4, is our primary technical result; it is used to derive our complexity trichotomy on ep-formulas from the known complexity trichotomy on pp-formulas (which was presented in Section 5.1.4).

Theorem 5.13. *(Equivalence theorem) Let Φ be a set of ep-formulas. There exists a set Φ^+ of prenex pp-formulas with the following property: the two counting case problems $\mathsf{count}[\Phi]$ and $\mathsf{count}[\Phi^+]$ are interreducible under counting slice reductions. In particular, there exists an algorithm that computes, given an ep-formula ϕ, a finite set ϕ^+ of prenex pp-formulas such that for any set Φ of ep-formulas, the set Φ^+ defined as $\bigcup\{\phi^+ \mid \phi \in \Phi\}$ has the presented property.*

We now state our trichotomy theorem on the complexity of counting answers to ep-formulas, and show how to prove it using the equivalence theorem.

Theorem 5.14. *(Trichotomy theorem) Let Φ be a computable set of ep-formulas of bounded arity, and let Φ^+ be the set of pp-formulas given by Theorem 5.13.*

1. *If Φ^+ satisfies the tractability condition, then it holds that* $\mathsf{param\text{-}count}[\Phi]$ *is in FPT.*
2. *If Φ^+ does not satisfy the tractability condition but satisfies the contraction condition, then it holds that* $\mathsf{param\text{-}count}[\Phi]$ *is interreducible with* p-clique *under counting FPT-reduction.*
3. *Otherwise, there is a counting FPT-reduction from the problem* p-$\#\mathsf{clique}$ *to* $\mathsf{param\text{-}count}[\Phi]$.

Proof. For (1), we use the counting slice reduction (U, r) from $\mathsf{count}[\Phi]$ to $\mathsf{count}[\Phi^+]$ given by Theorem 5.13. In particular, given as input $(\phi, \mathbf{B})$, it is first checked if $\phi \in \Phi$; if not, 0 is output. Otherwise, the algorithm for r is invoked on $(\phi, \phi^+, \mathbf{B})$, where ϕ^+ is as defined in the statement of Theorem 5.13; queries to $\mathsf{count}(\psi, \mathbf{B})$ where $\psi \in \Phi^+$ are resolved according to the algorithm of Theorem 5.11.

For (2) and (3), we make use of the result (Theorem 5.13) that the problems $\mathsf{count}[\Phi]$ and $\mathsf{count}[\Phi^+]$ are interreducible under counting slice reductions. For (2), we have from Theorem 5.12 that $\mathsf{count}[\Phi^+]$ and $\mathsf{clique}[\mathbb{N}]$ are interreducible under counting slice reductions. Hence, we obtain that the problems $\mathsf{clique}[\mathbb{N}]$ and $\mathsf{count}[\Phi]$ are interreducible under counting slice reductions, and the result follows from Theorem 5.9. For (3), we have from Theorem 5.12 that there is a counting slice reduction from $\#\mathsf{clique}[\mathbb{N}]$ to $\mathsf{count}[\Phi^+]$, and hence from $\#\mathsf{clique}[\mathbb{N}]$ to $\mathsf{count}[\Phi]$; the result then follows from Theorem 5.9. □

Let us remark that when case (2) applies, a consequence of this theorem is that the problem $\mathsf{param\text{-}count}[\Phi]$ is not in FPT unless W[1] is in FPT, since p-clique is W[1]-complete; in a similar fashion, when case (3) applies, the problem $\mathsf{param\text{-}count}[\Phi]$ is not in FPT unless ♯W[1] is in FPT, since p-$\#\mathsf{clique}$ is ♯W[1]-complete.

5.3. Examples

Before proving the equivalence theorem in full generality, we discuss some example ep-formulas to illustrate and preview some of the issues and difficulties with which the argument needs to cope.

Example 5.15. *Consider the formula*

$$\phi(w, x, y, z) := E(x, y) \wedge (E(w, x) \vee (E(y, z) \wedge E(z, z))).$$

As a first simplification step, we bring disjunction to the outermost level in ϕ:

$$\begin{aligned} &\phi(w, x, y, z) \\ &\equiv (E(x, y) \wedge E(w, x)) \vee (E(x, y) \wedge E(y, z) \wedge E(z, z)). \end{aligned}$$

Now let us set $\phi_1(w,x,y,z) \equiv E(x,y) \wedge E(w,x)$ *and also set* $\phi_2(w,x,y,z) \equiv E(x,y) \wedge E(y,z) \wedge E(z,z)$. *We can use inclusion-exclusion to count the number of satisfying assignments of* ϕ *on a structure* $\mathbf{B}$ *by*

$$|\phi(\mathbf{B})| = |\phi_1(\mathbf{B})| + |\phi_2(\mathbf{B})| - |(\phi_1 \wedge \phi_2)(\mathbf{B})|.$$

One point to observe is that, in this last expression, the count $|\phi_1(\mathbf{B})|$ *needs to be determined with respect to its set of liberal variabes* $\mathrm{lib}(\phi_1) = \{w,x,y,z\}$, *even though* z *does not appear in any atom of* ϕ_1. *If the count* $|\phi_1(\mathbf{B})|$ *is not computed in this way, the above expression for* $|\phi(\mathbf{B})|$ *fails to hold in general. The situation is analogous for the formula* ϕ_2, *where* w *does not appear in any atom.*

Example 5.16. *In general, if we are given an ep-formula* $\phi = \phi_1 \vee \ldots \vee \phi_n$ *where the* ϕ_i *are pp-formulas, then to compute the count* $|\phi(\mathbf{B})|$ *of* ϕ *relative to* $\mathbf{B}$, *it suffices to know the count for each of the* $2^n - 1$ *pp-formulas obtained by taking a conjunction of a non-empty subset of the* ϕ_i. *In this example, we will see that, in fact, one does not always need to consider all of these conjunctions. To this end, set* $V = \{w,x,y,z\}$ *and set*

$$\phi(V) = \phi_1(V) \vee \phi_2(V) \vee \phi_3(V)$$

where $\phi_1(V) = E(x,y) \wedge E(y,z)$, $\phi_2(V) = E(z,w) \wedge E(w,x)$ *and* $\phi_3(V) = E(w,x) \wedge E(x,y)$. *Applying inclusion-exclusion, we obtain*

$$\begin{aligned} |\phi(\mathbf{B})| =& |\phi_1(\mathbf{B})| + |\phi_2(\mathbf{B})| + |\phi_3(\mathbf{B})| \\ & - |(\phi_1 \wedge \phi_2)(\mathbf{B})| - |(\phi_1 \wedge \phi_3)(\mathbf{B})| \\ & - |(\phi_2 \wedge \phi_3)(\mathbf{B})| + |(\phi_1 \wedge \phi_2 \wedge \phi_3)(\mathbf{B})|. \end{aligned}$$

Now observe that the formulas ϕ_1, ϕ_2 *and* ϕ_3 *are actually equivalent to each other up to renaming variables; consequently, these formulas are equivalent in that, for any structure* $\mathbf{B}$, *they yield the same count:* $|\phi_1(\mathbf{B})| = |\phi_2(\mathbf{B})| = |\phi_3(\mathbf{B})|$. *In Section 5.4.1, we formalize and give a characterization of this notion of equivalence (on pp-formulas). The formulas* $\phi_1 \wedge \phi_3$ *and* $\phi_2 \wedge \phi_3$ *are also equivalent in this sense. We may thus obtain the following expression for* $|\phi(\mathbf{B})|$.

$$\begin{aligned} |\phi(\mathbf{B})| =& 3 \cdot |\phi_1(\mathbf{B})| - |(\phi_1 \wedge \phi_2)(\mathbf{B})| \\ & - 2 \cdot |(\phi_1 \wedge \phi_3)(\mathbf{B})| + |(\phi_1 \wedge \phi_2 \wedge \phi_3)(\mathbf{B})|. \end{aligned}$$

So far, we have only unified formulas that are equivalent up to renaming variables. In our parameterized complexity setting where ϕ *is the parameter, this does not yield a significant decrease in the complexity of computing* $|\phi(\mathbf{B})|$. *However, we will now observe a simplification that is more substantial in this sense. Namely, one can verify that the formulas* $\phi_1 \wedge \phi_2$ *and* $\phi_1 \wedge \phi_2 \wedge \phi_3$ *are identical. So, if we identify their terms in this last expression for* $|\phi(\mathbf{B})|$, *we obtain a cancellation and arrive to the following expression:*

$$|\phi(\mathbf{B})| = 3 \cdot |\phi_1(\mathbf{B})| - 2 \cdot |(\phi_1 \wedge \phi_3)(\mathbf{B})|.$$

The savings obtained by observing this cancellation are significant, in the following sense. The pp-formulas $\phi_1 \wedge \phi_2$ and $\phi_1 \wedge \phi_2 \wedge \phi_3$, which were cancelled, were the only formulas in the expression for $|\phi(\mathbf{B})|$ which did not have treewidth 1; they had treewidth 2. As it is known that the runtime of evaluation algorithms for quantifier-free pp-formulas scales with their treewidth [Mar10a], this reduction in treewidth yields a superior runtime for evaluating $|\phi(\mathbf{B})|$.

As we have seen in the above examples, counting on an ep-formula can, via inclusion-exclusion, reduce to counting on a finite set of pp-formulas. (This is carried out in our argument; see Section 5.4.3). As just seen in Example 5.16, there can be some subtlety in choosing a desirable set of pp-formulas to reduce to. One question not addressed so far is how one can reduce from counting on a such obtained set of pp-formulas to counting on the original ep-formula. To this end, let us revisit our first example.

Example 5.17. *Let us consider again the formulas of Example 5.15. Assume that we are given access to an oracle that lets us compute $|\phi(\mathbf{D})|$, for any structure $\mathbf{D}$ of our choice. We will see that, given a structure $\mathbf{B}$, we can compute $|\phi_1(\mathbf{B})|$, $|\phi_2(\mathbf{B})|$, and $|(\phi_1 \wedge \phi_2)(\mathbf{B})|$ efficiently using this oracle.*

To see this, consider the structure $\mathbf{C}$ with universe $C = \{1,2,3,4\}$ and $E^{\mathbf{C}} = \{(1,2),(2,3),(3,4),(4,4)\}$. It is easy to check that the formulas ϕ_1, ϕ_2 and $\phi_1 \wedge \phi_2$ all have a different number of answers with respect to $\mathbf{C}$. Now note that for every pp-formula ψ and every pair of structures $\mathbf{D}_1$, $\mathbf{D}_2$ we have $|\psi(\mathbf{D}_1 \times \mathbf{D}_2)| = |\psi(\mathbf{D}_1)| \cdot |\psi(\mathbf{D}_2)|$. Querying the oracle for $|\phi(\cdot)|$ on $\mathbf{B} \times \mathbf{C}^i$ for the values $i = 0,1,2$, we obtain the linear system

$$A \begin{pmatrix} |\phi_1(\mathbf{B})| \\ |\phi_2(\mathbf{B})| \\ -|(\phi_1 \wedge \phi_2)(\mathbf{B})| \end{pmatrix} = \begin{pmatrix} (\phi(\mathbf{B}) \\ \phi(\mathbf{B} \times \mathbf{C}) \\ \phi(\mathbf{B} \times \mathbf{C}^2) \end{pmatrix}$$

with

$$A = \begin{pmatrix} 1 & 1 & 1 \\ |\phi_1(\mathbf{C})| & |\phi_2(\mathbf{C})| & |(\phi_1 \wedge \phi_2)(\mathbf{C})| \\ |\phi_1(\mathbf{C})|^2 & |\phi_2(\mathbf{C})|^2 & |(\phi_1 \wedge \phi_2)(\mathbf{C})|^2 \end{pmatrix}.$$

Note that the entries of A can be computed efficiently, and the vector on the right-hand-side of the equation can be provided by our oracle. The matrix A is a Vandermonde matrix, as a consequence of the choice of $\mathbf{C}$. Thus, the system has a unique solution and can be solved to determine $|\phi_1(\mathbf{B})|$, $|\phi_2(\mathbf{B})|$, and $|(\phi_1 \wedge \phi_2)(\mathbf{B})|$, as desired.

In Example 5.17 we have seen that, for the particular ep-formula ϕ discussed, counting on ϕ is in a certain sense interreducible with counting on the pp-formulas

$$\{\phi_1, \phi_2, \phi_1 \wedge \phi_2\}.$$

The statement of the equivalence theorem (Theorem 5.13) asserts that for *any* ep-formula ϕ, there exists a finite set ϕ^+ of pp-formulas such that one has this interreducibility.

5.4. Proof of equivalence theorem

In this section, we give a decidable characterization of *counting equivalence* (Section 5.4.1); we then study a relaxation thereof which we call *semi-counting equivalence* (Section 5.4.2); we prove the equivalence theorem in the particular case of all-free ep-formulas (Section 5.4.3); and, we end by proving the equivalence theorem in its full generality (Section 5.4.4). Throughout this section, we generally assume pp-formulas to be prenex.

5.4.1. Counting equivalence

As we have seen in the examples of Section 5.3, it will be important to see when two different pp-formulas give same number of answers for every structure, because it will allow us to make simplifications in formulas we get by inclusion-exclusion. To this end, we make the following definition.

Definition 5.18. Define two fo-formulas $\phi(V), \phi'(V')$ to be *counting equivalent* if they are over the same vocabulary τ and for each finite τ-structure $\mathbf{B}$ it holds that $|\phi(\mathbf{B})| = |\phi'(\mathbf{B})|$.

In this subsection, we characterize counting equivalence for pp-formulas. To approach the characterization, we start off with an example.

Example 5.19. *It is apparent that logically equivalent formulas are counting equivalent, but the converse direction is not true. To see this, consider the pp-formulas* $\phi_1(x, y) = E(x, y)$ *and* $\phi_2(w, z) = E(w, z)$*. Obviously,* ϕ_1 *and* ϕ_2 *are counting equivalent (they just count the number of tuples in the relation* E *of a structure* $\mathbf{B}$*). But* ϕ_1 *and* ϕ_2 *are not logically equivalent; indeed, the assignments in* $\phi_1(\mathbf{B})$ *and* $\phi_2(\mathbf{B})$ *assign values to different variables.*

Note that one way of witnessing the counting equivalence of ϕ_1 *and* ϕ_2 *is simply renaming the variable* w *to* x *and* z *to* y *to get equivalent formulas. Since this syntactic renaming obviously does not change the number of satisfying assignments, one can conclude that* ϕ_1 *and* ϕ_2 *are counting equivalent.*

Example 5.19 motivates the following definition.

Definition 5.20. We say that two pp-formulas

$$(\mathbf{A}, S), (\mathbf{A}', S')$$

over the same signature are *renaming equivalent* if there exist surjections $h : S \to S'$ and $h' : S' \to S$ that can be extended to homomorphisms $\bar{h} : \mathbf{A} \to \mathbf{A}'$ and $\bar{h'} : \mathbf{A}' \to \mathbf{A}$, respectively.

Informally speaking, on pp-formulas, two formulas are renaming equivalent if they become logically equivalent after a renaming of variables, as occurred in Example 5.19. Hence, renaming equivalence is a relaxation of logical equivalence. Recall that logical equivalence of pp-formulas was characterized, in Theorem 5.3.

The main theorem of this subsection is that renaming equivalence does not only imply counting equivalence but is actually equivalent to it.

Theorem 5.21. *Two pp-formulas*

$$\phi_1(S_1), \phi_2(S_2)$$

are counting equivalent if and only if they are renaming equivalent.

Note that Theorem 5.21 gives a syntactic/algebraic characterization of counting equivalence which makes counting equivalence decidable by a straightforward algorithm and in fact even puts it into NP.

Before we prove Theorem 5.21, we start off with an simple observation that will be helpful in the proof.

Observation 5.22. *Let ϕ and ϕ' be counting equivalent pp-formulas. Then* $|\mathrm{lib}(\phi)| = |\mathrm{lib}(\phi')|$.

Proof. Let $\mathbf{C}$ be a structure that interprets every relation symbol in R of ϕ by $R^{\mathbf{C}} := \{0,1\}^{\mathsf{arity}(R)}$. Then $|\phi(\mathbf{C})| = 2^{|\mathrm{lib}(\phi)|}$ and $|\phi'(\mathbf{C})| = 2^{|\mathrm{lib}(\phi')|}$ and the claim follows directly. □

Proof. (Theorem 5.21) We begin with the backward direction; let $h_1 : S_1 \to S_2$ and $h_2 : S_2 \to S_1$ be the surjections from the definition of renaming equivalence. The existence of these surjections implies that $|S_1| = |S_2|$ and that each of h_1, h_2 is a bijection. Let $\mathbf{B}$ be an arbitrary structure. For each $f : S_2 \to B$ in $\phi_2(\mathbf{B})$, it is straightforward to verify that the composition $f(h_1)$ is in $\phi_1(\mathbf{B})$. Since the mapping that takes each such f to $f(h_1)$ is injective (due to h_1 being a bijection), we obtain that $|\phi_1(\mathbf{B})| \geq |\phi_2(\mathbf{B})|$. By symmetric reasoning, we can obtain that $|\phi_1(\mathbf{B})| \leq |\phi_2(\mathbf{B})|$, and we conclude that $|\phi_1(\mathbf{B})| = |\phi_2(\mathbf{B})|$.

For the other direction, let $\phi_1(S_1)$ and $\phi_2(S_2)$ be two pp-formulas over a common vocabulary τ that are not renaming equivalent; let $(\mathbf{A}_1, S_1)$ and $(\mathbf{A}_2, S_2)$ be the corresponding structures. By way of contradiction, assume that ϕ_1 and ϕ_2 are counting equivalent. If it holds that $|\mathrm{lib}(\phi_1)| \neq |\mathrm{lib}(\phi_2)|$, we are done by Observation 5.22. So we may assume, after potentially renaming some variables, that $\mathrm{lib}(\phi_1) = \mathrm{lib}(\phi_2) =: S$.

When $\mathbf{C}$, $\mathbf{D}$ are structures with $S \subseteq C \cap D$, let us define $\hom(\mathbf{C}, \mathbf{D}, S)$ to be the set of mappings from S to D that can be extended to a homomorphism from $\mathbf{C}$ to $\mathbf{D}$; denote by $\mathrm{surj}(\mathbf{C}, \mathbf{D}, S)$ the surjections $h : S \to S$ that lie in $\hom(\mathbf{C}, \mathbf{D}, S)$.

As $(\mathbf{A}_1, S_1)$ and $(\mathbf{A}_2, S_2)$ are by hypothesis not renaming equivalent, we may assume, without loss of generality, that $\mathrm{surj}(\mathbf{A}_1, \mathbf{A}_2, S) = \emptyset$. For $T \subseteq S$ let us use $\hom_T(\mathbf{A}_1, \mathbf{A}_2, S)$ to denote the set of mappings $h \in \hom(\mathbf{A}_1, \mathbf{A}_2, S)$ such that $h(S) \subseteq T$. By inclusion-exclusion we get

$$|\mathrm{surj}(\phi_1, \phi_2, S)| = \sum_{T \subseteq S} (-1)^{|S|-|T|} |\hom_T(\mathbf{A}_1, \mathbf{A}_2, S)|.$$

For $i \geq 0$ let $\hom_{i,T}(\mathbf{A}_1, \mathbf{A}_2, S)$ be the set of mappings $h \in \hom(\mathbf{A}_1, \mathbf{A}_2, S)$ such that h maps exactly i variables from S into T. Now for each $j = 1, \ldots, |S|$ we construct a new

structure $\mathbf{D}_{j,T}$ over the domain $D_{j,T}$. To this end, let $a^{(1)}, \ldots, a^{(j)}$ be copies of $a \in T$ that are not in A_2. Then we set

$$D_{j,T} := \{a^{(k)} \mid a \in A_2, a \in T, k \in [j]\} \cup (A_2 \setminus T).$$

We define a mapping $B : A_2 \to \mathcal{P}(D_{j,T})$, where $\mathcal{P}(D_{j,T})$ is the power set of $D_{j,T}$, by

$$B(a) := \begin{cases} \{a^{(k)} \mid k \in [j]\}\}, & \text{if } a \in T \\ \{a\}, & \text{otherwise.} \end{cases}$$

For every relation symbol $R \in \tau$ we define

$$R^{\mathbf{D}_{T,j}} := \bigcup_{(d_1,\ldots,d_s) \in R^{\mathbf{A}_2}} B(d_1) \times \ldots \times B(d_s).$$

Then every $h \in \hom_{i,T}(\mathbf{A}_1, \mathbf{A}_2, S)$ corresponds to j^i mappings in $\hom(\mathbf{A}_1, \mathbf{D}_{j,T}, S)$. Thus for each j we get

$$\sum_{i=1}^{|S|} j^i |\hom_{i,T}(\mathbf{A}_1, \mathbf{A}_2, S)| = |\hom(\mathbf{A}_1, \mathbf{D}_{j,T}, S)|.$$

This is a linear system of equations and the corresponding matrix is a Vandermonde matrix; consequently, the value $\hom_T(\mathbf{A}_1, \mathbf{A}_2, S) = \hom_{|S|,T}(\mathbf{A}_1, \mathbf{A}_2, S)$ can efficiently be computed from $|\hom(\mathbf{A}_1, \mathbf{D}, S)| = |\phi_1(\mathbf{D})|$ for some structures $\mathbf{D}$. We can similarly determine $|\hom_T(\mathbf{A}_2, \mathbf{D}, S)|$ as a function of $|\phi_2(\mathbf{D})|$ for the same structures $\mathbf{D}$. Since $|\phi_1(\mathbf{D})| = |\phi_2(\mathbf{D})|$ for every structure $\mathbf{D}$ by assumption, it follows that for every subset $T \subseteq S$ we have

$$|\hom_T(\mathbf{A}_1, \mathbf{A}_2, S)| = |\hom_T(\mathbf{A}_2, \mathbf{A}_2, S)|.$$

But then we have

$$|\mathrm{surj}(\mathbf{A}_1, \mathbf{A}_2, S)| = |\mathrm{surj}(\mathbf{A}_2, \mathbf{A}_2, S)|.$$

Since $\mathrm{surj}(\mathbf{A}_1, \mathbf{A}_2, S) = \emptyset$ and $\mathrm{id} \in \mathrm{surj}(\mathbf{A}_2, \mathbf{A}_2, S)$, this is a contradiction. Consequently, we obtain that ϕ_1 and ϕ_2 are not counting equivalent. □

5.4.2. Semi-counting equivalence

In this subsection, we study a relaxation of the notion of *counting equivalence*. This notion will be necessary when we emulate the approach of Example 5.17 in the proof of the Equivalence theorem: we will again construct a system of linear equations that we want to solve. In order to ensure solvability, we will make sure that the matrix of the system is again a Vandermonde matrix which in particular means that all its entries must be positive. Consequently, since the entries are of the form $|\phi(\mathbf{C})|^k$ for pp-formulas ϕ some carefully chosen structure $\mathbf{C}$ and integers k, it will be necessary to understand counting equivalence in the case where $\phi(\mathbf{C})$ is non-empty. The necessary notion is formalized by the following definition.

Definition 5.23. Call two prenex pp-formulas

$$\phi_1(V_1), \phi_2(V_2)$$

on the same vocabulary *semi-counting equivalent* if for each finite structure $\mathbf{B}$ such that $|\phi_1(\mathbf{B})| > 0$ and $|\phi_2(\mathbf{B})| > 0$, it holds that $|\phi_1(\mathbf{B})| = |\phi_2(\mathbf{B})|$.

Example 5.24. *The pp-formulas $\phi_1(x, y) = E(x, y)$ and $\phi_2(x, y) = \exists z(E(x, y) \wedge F(z))$ are not counting equivalent, because for every structure $\mathbf{B}$ for which $F^{\mathbf{B}} = \emptyset$, we have $|\phi_2(\mathbf{B})| = 0$ while $|\phi_1(\mathbf{B})|$ may be non-zero if $E^{\mathbf{B}}$ is non-empty. But if we have for a structure $\mathbf{B}$ such that $|\phi_2(\mathbf{B})| > 0$, then $F^{\mathbf{B}} \neq \emptyset$ and it is straightforward to verify that $|\phi_1(\mathbf{B})| = |\phi_2(\mathbf{B})|$. Consequently, we have that ϕ_1 and ϕ_2 are semi-counting equivalent.*

For each free prenex pp-formula $\phi(V)$, define $\widehat{\phi}(V)$ to be the pp-formula obtained from ϕ by removing each atom that occurs in a non-liberal component of ϕ (a component of ϕ not having liberal variables).

Example 5.25. *Consider the pp-formula ϕ discussed in Examples 5.2 and 5.4. This pp-formula has 4 components, namely, the pp-formulas $\psi_1(x, x')$, $\psi_2(y)$, $\psi_3(z)$, and $\psi_4(\emptyset)$ defined in Example 5.4. The formulas ψ_1, ψ_2, and ψ_3 are liberal, but the formula ψ_4 is not liberal. Recall that the formula $\phi(x, x', y, z)$ is equal to*

$$\exists y' \exists u \exists v \exists w(E(x, x') \wedge E(y, y') \wedge F(u, v) \wedge G(u, w))$$

and that we have $\psi_4(\emptyset) = \exists u \exists v \exists w(F(u, v) \wedge G(u, w))$. We hence have that $\widehat{\phi}(x, x', y, z)$ is the formula

$$\exists y' \exists u \exists v \exists w(E(x, x') \wedge E(y, y')).$$

The following characterization of semi-counting equivalence is the main theorem of this subsection.

Theorem 5.26. *Let $\phi_1(V_1), \phi_2(V_2)$ be two free prenex pp-formulas. It holds that $\phi_1(V_1)$ and $\phi_2(V_2)$ are semi-counting equivalent if and only if $\widehat{\phi_1}(V_1)$ and $\widehat{\phi_2}(V_2)$ are counting equivalent.*

We will use the following proposition in the proof of Theorem 5.26.

Proposition 5.27. *Let $\phi(V)$ be a free prenex pp-formula. Then for every structure $\mathbf{B}$ we have $\phi(\mathbf{B}) = \emptyset$ or $\phi(\mathbf{B}) = \widehat{\phi}(\mathbf{B})$.*

Proof. Let $\mathbf{B}$ be a structure. Let ψ be the conjunction of the components deleted from ϕ to obtain $\widehat{\phi}$. If ψ is false on $\mathbf{B}$, then obviously $\phi(\mathbf{B}) = \emptyset$. Otherwise, ψ is true on $\mathbf{B}$, and for any assignment $f : V \to B$, it holds that $\mathbf{B}, f \models \phi$ if and only if $\mathbf{B}, f \models \widehat{\phi}$. □

Proof. (Theorem 5.26) Assume first that $\widehat{\phi_1}$ and $\widehat{\phi_2}$ are counting equivalent. Let $\mathbf{B}$ be a structure. Then if $|\phi_1(\mathbf{B})| > 0$ and $|\phi_2(\mathbf{B})| > 0$, we have by Proposition 5.27 and counting equivalence of $\widehat{\phi_1}$ and $\widehat{\phi_2}$ that $|\phi_1(\mathbf{B})| = |\widehat{\phi_1}(\mathbf{B})| = |\widehat{\phi_2}(\mathbf{B})| = |\phi_2(\mathbf{B})|$, so ϕ_1 and ϕ_2 are semi-counting equivalent.

For the other direction let now ϕ_1 and ϕ_2 be semi-counting equivalent. By way of contradiction, we assume that $\widehat{\phi_1}$ and $\widehat{\phi_2}$ are not counting equivalent. Then by definition there is a structure $\mathbf{B}$ such that $|\widehat{\phi_1}(\mathbf{B})| \neq |\widehat{\phi_2}(\mathbf{B})|$. Note that each component of $\widehat{\phi_1}$ and $\widehat{\phi_2}$ has a liberal variable.

Let $\mathbf{I} = \mathbf{I}_\tau$, where τ is the vocabulary of ϕ_1 and ϕ_2. For each $k \in \mathbb{N}$ we denote by $\mathbf{B} + k\mathbf{I}$ the structure we get from $\mathbf{B}$ by disjoint union with k copies of $\mathbf{I}$. Note that for $k > 0$ we have $|\phi(\mathbf{B} + k\mathbf{I})| > 0$ for every pp-formula ϕ. Consequently, for every $k > 0$ we have $|\phi_1(\mathbf{B} + k\mathbf{I})| = |\widehat{\phi_1}(\mathbf{B} + k\mathbf{I})|$ and $|\phi_2(\mathbf{B} + k\mathbf{I})| = |\widehat{\phi_2}(\mathbf{B} + k\mathbf{I})|$ by Proposition 5.27. By the semi-counting equivalence of ϕ_1 and ϕ_2 we also have $|\phi_1(\mathbf{B} + k\mathbf{I})| = |\phi_2(\mathbf{B} + k\mathbf{I})|$ for all $k > 0$. It follows that $|\widehat{\phi_1}(\mathbf{B} + k\mathbf{I})| = |\widehat{\phi_2}(\mathbf{B} + k\mathbf{I})|$ for $k > 0$.

Let $\phi_{1,1}, \ldots, \phi_{1,n}$ denote the components of $\widehat{\phi_1}$, and let $\phi_{2,1}, \ldots, \phi_{2,m}$ denote the components of $\widehat{\phi_2}$. Because every component of $\widehat{\phi_1}$ has a liberal variable, we have

$$\begin{aligned} |\widehat{\phi_1}(\mathbf{B} + k\mathbf{I})| &= \sum_{J \subseteq [n]} k^{n-|J|} \prod_{j \in J} |\phi_{1,j}(\mathbf{B})| \\ &= \sum_{\ell=0}^{n} k^{n-\ell} \sum_{J \subseteq [n], |J|=\ell} \prod_{j \in J} |\phi_{1,j}(\mathbf{B})|. \end{aligned}$$

We can express $|\widehat{\phi_2}(\mathbf{B} + k\mathbf{I})|$ analogously. The expressions are polynomials in k and they are equal for every positive integer k by the observations above; thus the coefficients of the polynomials must coincide. The coefficients of k^0, namely the values $\prod_{j \in [n]} |\phi_{1,j}(\mathbf{B})|$ and $\prod_{j \in [m]} |\phi_{2,j}(\mathbf{B})|$, are thus equal. But then we get

$$|\widehat{\phi_1}(\mathbf{B})| = \prod_{j \in [n]} |\phi_{1,j}(\mathbf{B})| = \prod_{j \in [m]} |\phi_{2,j}(\mathbf{B})| = |\widehat{\phi_2}(\mathbf{B})|,$$

which is a contradiction to our assumption. □

Corollary 5.28. *Semi-counting equivalence is an equivalence relation (on pp-formulas).*

We now present a lemma that will be of utility; it is proved by induction.

Lemma 5.29. *Let $\phi_1(S_1), \ldots, \phi_n(S_n)$ be pp-formulas over the same vocabulary τ, which are liberal (that is, with each $|S_i| > 0$). Then there is a structure $\mathbf{C}$ (over τ) such that*

- *for all pp-formulas ϕ (over τ) it holds that $|\phi(\mathbf{C})| > 0$, and*
- *for all $i, j \in [n]$ such that ϕ_i and ϕ_j are not semi-counting equivalent, it holds that $|\phi_i(\mathbf{C})| \neq |\phi_j(\mathbf{C})|$.*

In order to establish this lemma, we first prove the following lemma.

Lemma 5.30. *Let $\phi_1(S_1)$ and $\phi_2(S_2)$ be two pp-formulas over a vocabulary τ that are not semi-counting equivalent. Then there is a structure $\mathbf{D}$ such that for every primitive positive formula ϕ over τ we have $|\phi(\mathbf{D})| > 0$ and $|\phi_1(\mathbf{D})| \neq |\phi_2(\mathbf{D})|$.*

Proof. Let $\mathbf{B}$ be any structure on which ϕ_1 and ϕ_2 have a non-zero but different number of solutions. Such a structure exists by definition of semi-counting equivalence. We claim that we can choose $\mathbf{D} = \mathbf{B} + k\mathbf{I}$ for some $k \in \mathbb{N}, k > 0$ where $\mathbf{B} + k\mathbf{I}$ is defined as in the proof of Theorem 5.26. By way of contradiction, assume that $|\phi_1(\mathbf{B}+k\mathbf{I})| = |\phi_2(\mathbf{B}+k\mathbf{I})|$ for all $k \in \mathbb{N}, k > 0$. Then with the same argument as in the proof of Theorem 5.26 we get the contradiction that $|\phi_1(\mathbf{B})| = |\phi_2(\mathbf{B})|$. □

Proof. (Lemma 5.29) We prove this by induction on n; the case $n = 2$ is implied by Lemma 5.30.

When $n > 2$, we first observe that it suffices to prove the result when the ϕ_i are pairwise not semi-counting equivalent, so we assume that this holds. Let $\mathbf{D}$ be the structure that we get by induction for $\phi_1, \ldots, \phi_{n-1}$. We may assume w.l.o.g. that

$$|\phi_1(\mathbf{D})| < |\phi_2(\mathbf{D})| < \ldots < |\phi_{n-1}(\mathbf{D})|.$$

If it holds that $|\phi_n(\mathbf{D})| \neq |\phi_i(\mathbf{D})|$ for every $i \in [n-1]$, then we are done. So we assume that there is an index $i \in [n-1]$ such that $|\phi_n(\mathbf{D})| = |\phi_i(\mathbf{D})|$.

Let $\mathbf{D}'$ be the structure we get by applying Lemma 5.30 to ϕ_n and ϕ_i.

Now choose k such that for every j with $1 < j \leq i$ we have

$$\frac{|\phi_{j-1}(\mathbf{D})|^k}{|\phi_j(\mathbf{D})|^k} < \frac{1}{|\mathrm{lib}(\phi_{j-1})|^{|D'|}}.$$

Then we have for every $\ell \geq k$ and $1 < j < i$

$$\begin{aligned}
|\phi_{j-1}(\mathbf{D}^\ell \times \mathbf{D}')| &= |\phi_{j-1}(\mathbf{D}^\ell)| \cdot |\phi_{j-1}(\mathbf{D}')| \\
&\leq |\phi_{j-1}(\mathbf{D}^\ell)| \cdot |\mathrm{lib}(\phi_{j-1})|^{|D'|} \\
&< |\phi_j(\mathbf{D}^\ell)| \\
&\leq |\phi_j(\mathbf{D}^\ell)| \cdot |\phi_j(\mathbf{D}')| \\
&= |\phi_j(\mathbf{D}^\ell \times \mathbf{D}')|.
\end{aligned}$$

Analogously, we get for every $\ell > k$ that

$$|\phi_{i-1}(\mathbf{D}^\ell \times \mathbf{D}')| < |\phi_n(\mathbf{D}^\ell \times \mathbf{D}')|.$$

Now choose k' such that for every j with $i \leq j < n$ we have

$$\frac{|\phi_{j+1}(\mathbf{D})|^{k'}}{|\phi_j(\mathbf{D})|^{k'}} > |\mathrm{lib}(\phi_j)|^{|D'|}.$$

Then we have for every $\ell > k'$ and every $i \leq j < n$

$$\begin{aligned}
|\phi_j(\mathbf{D}^\ell \times \mathbf{D}')| &= |\phi_j(\mathbf{D}^\ell)| \cdot |\phi_j(\mathbf{D}')| \\
&\leq |\phi_j(\mathbf{D}^\ell)| \cdot |\mathrm{lib}(\phi_j)|^{|D'|} \\
&< |\phi_{j+1}(\mathbf{D}^\ell)| \\
&\leq |\phi_{j+1}(\mathbf{D}^\ell)| \cdot |\phi_j(\mathbf{D}')| \\
&= |\phi_{j+1}(\mathbf{D}^\ell \times \mathbf{D}')|.
\end{aligned}$$

Similarly, we get for every $\ell > k$ that

$$|\phi_{i+1}(\mathbf{D}^\ell \times \mathbf{D}')| > |\phi_n(\mathbf{D}^\ell \times \mathbf{D}')|.$$

Now choosing $\ell = \max(k, k')$ and noting that

$$\begin{aligned} |\phi_i(\mathbf{D}^\ell \times \mathbf{D}')| &= |\phi_i(\mathbf{D}^\ell)| \cdot |\phi_i(\mathbf{D}')| \\ &\neq |\phi_n(\mathbf{D}^\ell)| \cdot |\phi_n(\mathbf{D}')| \\ &= |\phi_n(\mathbf{D}^\ell \times \mathbf{D}')| \end{aligned}$$

completes the proof with $\mathbf{C} = \mathbf{D}^\ell \times \mathbf{D}'$. □

The following is a consequence of this lemma.

Lemma 5.31. *Let $\phi_1(S_1), \ldots, \phi_n(S_n)$ be connected, liberal pp-formulas over the same vocabulary τ that are pairwise not counting equivalent. Then there exists a structure $\mathbf{C}$ (over τ) such that*

- *for all pp-formulas ϕ (over τ) it holds that $|\phi(\mathbf{C})| > 0$, and*
- *for all distinct $i, j \in [n]$, it holds that $|\phi_i(\mathbf{C})| \neq |\phi_j(\mathbf{C})|$.*

Proof. By Lemma 5.29, it suffices to show that the pp-formulas ϕ_i are pairwise not semi-counting equivalent. Since each ϕ_i is connected and liberal, we have $\phi_i = \widehat{\phi_i}$. Thus, by the hypothesis that the ϕ_i are pairwise not counting equivalent in combination with Theorem 5.26, we obtain that the ϕ_i are pairwise not semi-counting equivalent. □

5.4.3. The all-free case

The aim of this subsection is the proof of Theorem 5.13 in the special case of all-free ep-formulas. Recall that an ep-formula is *all-free* if it is the disjunction of prenex pp-formulas, each of which is *free* in that it has a non-empty set of free variables. We will later in Section 5.4.4 use the result on all-free formulas to prove the general version of Theorem 5.13.

For every $\phi(V) \in \Phi$ we define a set ϕ^* of free pp-formulas; then, we define $\Phi^* = \bigcup_{\phi \in \Phi} \phi^*(V)$. Let $\phi(V) = \phi_1(V) \vee \ldots \vee \phi_s(V)$ where the $\phi_i(V)$ are free pp-formulas. By inclusion-exclusion we have for every structure $\mathbf{B}$ that

$$\begin{aligned} |\phi(\mathbf{B})| &= \sum_{J \in [s]} (-1)^{|J|+1} |(\bigwedge_{j \in J} \phi_j)(\mathbf{B})| \\ &= \sum_{J \in [s]} (-1)^{|J|+1} |\phi_J(\mathbf{B})|, \end{aligned} \tag{5.1}$$

where the $\phi_J(V) = \bigwedge_{j \in J} \phi_j(V)$ are pp-formulas. Now iteratively do the following: If there are two summands $c|\phi_J(\mathbf{B})|$ and $c'|\phi_{J'}(\mathbf{B})|$ such that ϕ_J and $\phi_{J'}$ are counting equivalent, delete both summands and add $(c + c')|\phi_J|$ to the sum. When this operation can no longer be applied, delete all summands with coefficient zero. The pp-formulas that remain in the sum form the set ϕ^*.

Example 5.32. *It shall be advantageous to again consider Example 5.16. There we started off with*

$$\phi(V) = \phi_1(V) \vee \phi_2(V) \vee \phi_3(V).$$

Inclusion-exclusion yields

$$\begin{aligned}|\phi(\mathbf{B})| =& |\phi_1(\mathbf{B})| + |\phi_2(\mathbf{B})| + |\phi_3(\mathbf{B})| \\ & - |(\phi_1 \wedge \phi_2)(\mathbf{B})| - |(\phi_1 \wedge \phi_3)(\mathbf{B})| \\ & - |(\phi_2 \wedge \phi_3)(\mathbf{B})| + |(\phi_1 \wedge \phi_2 \wedge \phi_3)(\mathbf{B})|.\end{aligned}$$

Now we simplify as described above and get

$$|\phi(\mathbf{B})| = 3 \cdot |\phi_1(\mathbf{B})| - 2 \cdot |(\phi_1 \wedge \phi_3)(\mathbf{B})|.$$

Consequently, for this example we have

$$\phi^* = \{\phi_1, \phi_1 \wedge \phi_3\}.$$

The algorithm discussed above directly yields the following proposition.

Proposition 5.33. *There exists an algorithm that, when an all-free ep-formula ϕ is given as input, outputs a set $\phi^* := \{\phi_1^*, \ldots, \phi_\ell^*\}$ of free pp-formulas, which are pairwise not counting equivalent, and coefficients $c_1, \ldots, c_\ell \in \mathbb{Z} \setminus \{0\}$ such that for every structure $\mathbf{B}$,*

$$|\phi(\mathbf{B})| = \sum_{i=1}^{\ell} c_i |\phi_i^*(\mathbf{B})|.$$

We will also require the following two facts for our proof.

Proposition 5.34. *Let us presume that $\phi(S)$ and $\phi'(S')$ are two semi-counting equivalent free pp-formulas that are not counting equivalent and let $(\mathbf{A}, S)$ and $(\mathbf{A}', S')$ be the structures of ϕ and ϕ', respectively. Then $\mathbf{A}$ and $\mathbf{A}'$ are not homomorphically equivalent.*

Proof. $\phi(S)$ and $\phi'(S')$ are semi-counting equivalent, so we have by Theorem 5.26 and Theorem 5.21 that $\widehat{\phi(S)}$ and $\widehat{\phi'(S')}$ are renaming equivalent. It follows that $\mathbf{A}$ and $\mathbf{A}'$ are homomorphically equivalent via homomorphisms $h : A \to A'$, $h' : A' \to A$ that act as bijections between S and S'.

If there exists a homomorphism g from $\mathbf{A}$ to $\mathbf{A}'$, then we can extend h (using the definition of g) to be defined on the components of ϕ deleted in the construction of $\widehat{\phi}$, to obtain a homomorphism from $\mathbf{A}$ to $\mathbf{A}'$ extending h. If there exists a homomorphism g' from $\mathbf{A}'$ to $\mathbf{A}$, we can extend h' in an analogous way. However, the existence of both such extensions would imply by definition that $\phi(S)$ and $\phi'(S')$ are counting equivalent. We may thus conclude that either there is no homomorphism $\mathbf{A} \to \mathbf{A}'$ or there is no homomorphism $\mathbf{A}' \to \mathbf{A}$. □

Lemma 5.35. *There is an oracle FPT-algorithm that performs the following: given a set $\phi_1, \dots, \phi_s$ of semi-counting equivalent free pp-formulas that are pairwise not counting equivalent, a sequence $c_1, \dots, c_s \in \mathbb{Z} \setminus \{0\}$, and a structure $\mathbf{B}$, the algorithm computes $|\phi_i(\mathbf{B})|$ for every $i \in [s]$; it may make calls to an oracle that provides $\sum_{i=1}^{s} c_i \cdot |\phi_i(\mathbf{B}')|$ upon being given a structure $\mathbf{B}'$. Here, the ϕ_i with the c_i constitute the parameter.*

To establish this lemma, we first demonstrate the following proposition.

Proposition 5.36. *Let $\phi_1, \dots, \phi_s$ be a sequence of semi-counting equivalent pp-formulas that are pairwise not counting equivalent. Then there is a structure $\mathbf{C}$ and $i \in [s]$ such that $\mathbf{C} \models \phi_i$ but $\mathbf{C} \not\models \phi_j$ for all $j \in [s] \setminus \{i\}$.*

Proof. Let $\mathbf{A}_1, \dots, \mathbf{A}_n$ be the structures of the queries $\phi_1, \dots, \phi_n$. By Proposition 5.34 the structures $\mathbf{A}_i$ are pairwise not homomorphically equivalent. For $i, j \in [n]$, we write $\phi_i < \phi_j$ if there is a homomorphism from $\mathbf{A}_i$ to $\mathbf{A}_j$. It is easy to check that $<$ induces a partial order on the ϕ_i. Let ϕ_i be a minimal element of this partial order, then there is no homomorphism from any $\mathbf{A}_j$ to ϕ_i with $i \neq j$. Setting $\mathbf{C} = \mathbf{A}_i$ completes the proof. □

Proof. (Lemma 5.35) We give and algorithm that recursively computes the $|\phi_i(\mathbf{B})|$ one after the other. So let the parameter and the input be given as in the statement of the lemma. By Proposition 5.36, there is an $i \in [n]$ and a structure $\mathbf{C}$ such that $\mathbf{C} \models \phi_i$ but $\mathbf{C} \not\models \phi_j$ for all $j \in [s] \setminus \{i\}$. W.l.o.g. assume $i = s$. Then $|\phi_i(\mathbf{B} \times \mathbf{C})| = 0$ for $i < s$. Consequently, we have that the oracle lets us compute $c_s \cdot |\phi_n(\mathbf{B} \times \mathbf{C})| = c_s \cdot |\phi_n(\mathbf{B})| \cdot |\phi_n(\mathbf{C})|$. Computing $|\phi_n(\mathbf{C})|$ by brute force then yields $|\phi_s(\mathbf{B})|$.

Now note that for every structure $\mathbf{B}'$ we can also compute $\sum_{i=1}^{s-1} c_i \cdot |\phi_i(\mathbf{B}')|$ by this approach with one subtraction. So we can apply the algorithm again for $\phi_1, \dots, \phi_{s-1}$, answering oracle queries for the smaller sum $\sum_{i=1}^{s-1} c_i \cdot |\phi_i(\mathbf{B}')|$ with the help of the oracle for $\sum_{i=1}^{s} c_i \cdot |\phi_i(\mathbf{B}')|$. □

We can now prove Theorem 5.13 for all-free ep-formulas.

Theorem 5.37. *Let Φ be a set of all-free ep-formulas. There exists a set Φ^* of free prenex pp-formulas such that the counting case problems $\mathsf{count}[\Phi]$ and $\mathsf{count}[\Phi^*]$ are equivalent under counting slice reductions.*

Before giving the technical details of the proof of Theorem 5.37, let us first describe the ideas. The proof follows the approach presented in the examples of Section 5.3. In particular, the less straightforward reduction from $\mathsf{count}[\Phi^*]$ to $\mathsf{count}[\Phi]$ proceeds as we did in Example 5.17. Given ϕ and ϕ^*, we can evaluate $|\phi'(\mathbf{B})|$ for $\phi' \in \phi^*$ with an oracle for $|\phi(\mathbf{B} \times \mathbf{C}^\ell)|$ for a suitable structure $\mathbf{C}$ as in that example. The main difference is that, instead of having $\mathbf{C}$ explicitly as in Example 5.17, we here know from Lemma 5.29 that a structure $\mathbf{C}$ exists for which all formulas in ϕ^* have a different number of satisfying assignments. We can then compute $\mathbf{C}$ by brute force as it depends only on ϕ. This then allows to compute $\phi'(\mathbf{B})$ by solving a system of linear equations.

We now give the technical detail of the proof.

Proof. Let us first specify the reduction from $\mathsf{count}[\Phi]$ to $\mathsf{count}[\Phi^*]$, which is quite straightforward. The relation U is the set of pairs (ϕ, ϕ^*) such that ϕ is an all-free ep-formula and ϕ^* is the output of the algorithm of Proposition 5.33 on input ϕ. Obviously, this satisfies the coverage condition. Then the oracle-FPT-algorithm to compute $\phi(\mathbf{B})$ given ϕ, ϕ^* and $\mathbf{B}$ first computes all of the $|\phi_i^*(\mathbf{B})|$ by oracle calls and then uses Proposition 5.33. This completes the reduction.

For the other direction, let $\phi' \in \Phi^*$. We set U to be the set of all pairs $(\phi', \{\phi\})$ such that ϕ is an all-free ep-formula and $\phi' \in \phi^*$. Given ϕ', ϕ and $\mathbf{B}$, we compute $|\phi'(\mathbf{B})| := r(\phi', \{\phi\}, \mathbf{B})$ as follows: Let $\phi_1^*, \ldots, \phi_s^*$ be the equivalence classes of ϕ^* with respect to semi-counting equivalence. Now choose a strucuture $\mathbf{C}$ as in Lemma 5.29. Then for $\psi, \psi' \in \phi^*$ we have $|\psi(\mathbf{C})| \neq |\psi'(\mathbf{C})|$ if ψ and ψ' are from different equivalence classes with respect to semi-counting equivalence, and otherwise $|\psi(\mathbf{C})| = |\psi'(\mathbf{C})| > 0$. Fix for each $j \in [s]$ a formula in ϕ_j^* and call it ψ_j. Moreover, denote by c_ψ the coefficiencent of ψ in Proposition 5.33. Using this notation and Proposition 5.33 we obtain, for every $\ell \in \mathbb{N}$, that

$$|\phi(\mathbf{B} \times \mathbf{C}^\ell)| = \sum_{j=1}^{s} |\psi_j(\mathbf{C})|^\ell \Big(\sum_{\psi \in \phi_j^*} c_\psi |\psi(\mathbf{B})| \Big).$$

Note that this is a linear equation where the coefficients have the form $|\psi_j(\mathbf{C})|^\ell$; these can be computed by brute force. Letting ℓ range from 0 to $s-1$ thus yields a system of linear equations whose coefficient matrix is a Vandermonde matrix. Consequently, with s oracle calls we can compute $\sum_{\psi \in \phi_j^*} c_\psi |\psi(\mathbf{B})|$ for each j. We use Lemma 5.35 to compute $\phi'(\mathbf{B})$. □

5.4.4. The general case

We now indicate how to prove Theorem 5.13 in its full generality.

We may assume that each ep-formula $\phi \in \Phi$ is normalized. For each ep-formula ϕ, define ϕ_{af} to be the all-free part of ϕ, that is, the disjunction of the ϕ-disjuncts that are free; define Φ_{af} to be $\{\phi_{\mathrm{af}} \mid \phi \in \Phi\}$; and, define ϕ_{af}^- to be the set of formulas in ϕ_{af}^* that do not logically entail a sentence disjunct of ϕ. We define ϕ^+ to be the union of ϕ_{af}^- and the set containing each pp-sentence disjunct of ϕ; and, we define Φ^+ to be $\bigcup_{\phi \in \Phi} \phi^+$.

Example 5.38. *Set $V = \{w, x, y, z\}$; we consider the formulas $\phi(V) = \phi_1(V) \vee \phi_2(V) \vee \phi_3(V)$ defined in Example 5.16. Define*

$$\theta_1(V) = \exists a \exists b \exists c \exists d E(a, b) \wedge E(b, c) \wedge E(c, d),$$

and define

$$\theta(V) = \phi_1(V) \vee \phi_2(V) \vee \phi_3(V) \vee \theta_1(V).$$

The all-free part of θ is $\theta_{\mathrm{af}} = \phi_1(V) \vee \phi_2(V) \vee \phi_3(V)$, since each of these three disjuncts has a non-empty set of free variables, whereas θ_1 has an empty set of free variables. According to Example 5.32, we have

$$\theta_{\mathrm{af}}^* = \{\phi_1, \phi_1 \wedge \phi_3\}.$$

Now, observe that $\phi_1 \wedge \phi_3$ logically entails the sentence disjunct θ_1 of θ; on the other hand, ϕ_1 does not logically entail θ_1. Hence, we have that $\theta^-_{\mathrm{af}} = \{\phi_1\}$. We have θ^+ to be the union of θ^-_{af} and $\{\theta_1\}$, so

$$\theta^+ = \{\phi_1, \theta_1\}.$$

The idea of the proof of Theorem 5.13 is as follows. The counting slice reduction from $\mathsf{count}[\Phi]$ to $\mathsf{count}[\Phi^+]$ has U as the set of pairs (ϕ, ϕ^+) where ϕ is a normalized ep-formula; r on $(\phi(V), \phi^+, \mathbf{B})$ behaves as follows. First, it checks if there is a sentence disjunct θ of ϕ that is true on $\mathbf{B}$; if so, it outputs $|B|^{|V|}$; otherwise, it makes use of the counting slice reduction from $\mathsf{count}[\Phi_{\mathrm{af}}]$ to $\mathsf{count}[\Phi^*_{\mathrm{af}}]$. The counting slice reduction from $\mathsf{count}[\Phi^+]$ to $\mathsf{count}[\Phi]$ has U as the set $\{(\psi, \{\phi\}) \mid \psi \in \phi^+\}$; r on $(\psi(V), \phi(V), \mathbf{B})$ is defined as follows. When $\psi \in \phi^-_{\mathrm{af}}$, the counting slice reduction (U', r') from $\mathsf{count}[\Phi^*_{\mathrm{af}}]$ to $\mathsf{count}[\Phi_{\mathrm{af}}]$ is used to determine $|\psi(\mathbf{B})|$; this is performed by passing to r' a treated version of $\mathbf{B}$, on which no sentence disjunct of ϕ may hold. When ψ is a sentence disjunct of ϕ, an oracle query is made to obtain the count of ϕ on a treated version of $\mathbf{B}$; on this treated version, it is proved that all assignments satisfy ϕ if and only if $\mathbf{B} \models \psi$.

5.5. Conclusion

We have shown a trichotomy for the parameterized complexity of counting satisfying assignments to existential positive formulas of bounded arity. To this end, the main technical contribution was the equivalence theorem (Theorem 5.13) stating that for every set of existential positive formulas there is a set of primitive positive formulas that is computationally equivalent with respect to the counting problem studied. After showing this equivalence theorem, we could derive our trichotomy in a rather straightforward fashion by invoking a previous trichotomy for primitive positive formulas as a black-box.

In order to prove the equivalence theorem, we gave a syntactic characterization for when two pp-formulas are counting equivalent, that is, have the same number of satisfying assignments with respect to every finite structure. This result can be seen as an adaption, to the counting setting, of classical work of Chandra and Merlin [CM77] that characterizes logical equivalence of primitive positive formulas.

Let us note that the assumption of bounded arity is not needed in the proof of the equivalence theorem. It only appears in our trichotomy theorem because it is already present in the previous trichotomy on primitive positive formulas that we use. Consequently, if one could adapt the work of Marx [Mar10b] on model checking unbounded arity primitive positive formulas to counting to show a dichotomy or trichotomy for counting, this would directly give the corresponding result for existential positive formulas by applying our equivalence theorem.

Finally, let us remark that we are not aware of any fragment of first-order logic extending existential positive queries for which even model checking is understood, from the viewpoint of classifying the complexity of all sets of queries (for more information, see the discussion in the introduction of the article [Che14b]). Hence, the research project of extending our complexity classification beyond existential positive queries would first require an advance in the study of model checking in first-order logic.

6. DNNF Lower Bounds

This chapter presents the results of [27] on how to prove lower bounds for DNNF and their subclasses by making the connection to multi-partition communication complexity. See Chapter 2 for a longer introduction giving context and related work.

After some preliminaries in Section 6.1, we will first introduce the technique for the basic framework for general DNNF in Section 6.2. Then we will show in Section 6.3 how to use it to show some separations left open in the original knowledge compilation map paper [DM02]. Finally, in Section 6.4 we show how the lower bound techniques for structured DNNF in [PD10] fit into our framework as a special case and prove a conjecture from that paper.

6.1. Preliminaries

In this section, we introduce the notions of decomposable circuits (from knowledge compilation) and rectangle covers (from communication complexity).

Decomposable NNFs (DNNFs). We consider circuits in negation normal form, in short NNFs, which are (Boolean) circuits over fanin 2 conjunction and disjunction gates, labelled with $\wedge$ and $\vee$, whose inputs are labeled by literals.[1] The size of an NNF C, denoted by $|C|$, is the number of its gates.[2]

Let X be a finite set of variables. An NNF C over X is an NNF whose input gates are labelled with literals over variables in X. The (Boolean) function $f_C \colon \{0,1\}^X \to \{0,1\}$ computed by an NNF C over X is defined in the usual way. We let $\mathsf{sat}(C) = \mathsf{sat}(f_C) = f_C^{-1}(1)$ denote the set of satisfying assignments of C and f_C. Two NNFs C and C' over X are *equivalent* if $\mathsf{sat}(C) = \mathsf{sat}(C')$; if C and C' are equivalent, we write $C \equiv C'$. We also write $C \equiv f$ if $f_C = f$.

For a gate g in an NNF C over X, we let C_g denote the subcircuit of C rooted at g. In particular, $C_g = C$ if g is the output gate of C. For an NNF C over variables X and a gate $g \in C$, we let $\mathsf{var}(C_g) \subseteq X$ denote the variables appearing at input gates of C_g.

Let g be an $\wedge$-gate in an NNF C, and let h and h' be two distinct gates wiring g in C. Then g is called *decomposable* if $\mathsf{var}(C_h) \cap \mathsf{var}(C_{h'}) = \emptyset$. The decomposability of g in C implies that the two subcircuits C_h and $C_{h'}$ are "independent", in the sense that

$$|\mathsf{sat}(C_h \wedge C_{h'})| = |\mathsf{sat}(C_h)| \cdot |\mathsf{sat}(C_{h'})|,$$

[1] For technical convenience we admit circuits formed by a single gate labelled with 0 or 1, but assume that circuits with at least two gates do not contain constants.

[2] $|C| > 0$, as C contains at least the output gate. Circuits over unbounded fanin conjunctions and disjunctions can be quadratically simulated by fanin 2 circuits.

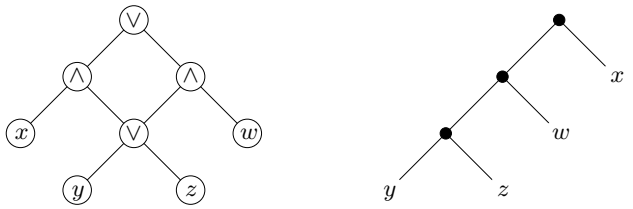


Figure 6.1.: A DNNF (left) and a vtree (right). The DNNF is structured (it respects the vtree), but not deterministic.

when viewing each circuit involved in the equation as an NNF over its own variables. An NNF whose $\wedge$-gates are decomposable is called a *decomposable* NNF (short, *DNNF*).

Let g be an $\vee$-gate in an NNF C, and let h and h' be two distinct gates wiring g in C. Then g is called *deterministic* if $\mathsf{sat}(C_h) \cap \mathsf{sat}(C_{h'}) = \emptyset$, viewing each circuit involved in the equation as an NNF over $\mathsf{var}(C)$. The determinism of g in C implies that the two subcircuits C_h and $C_{h'}$ are "independent", in the sense that

$$|\mathsf{sat}(C_h \vee C_{h'})| = |\mathsf{sat}(C_h)| + |\mathsf{sat}(C_{h'})|,$$

when viewing each circuit involved in the equation as an NNF over $\mathsf{var}(C_h) \cup \mathsf{var}(C_{h'})$. An NNF whose $\vee$-gates are all deterministic is called a *deterministic* NNF.

Let Y be a finite nonempty set of variables. A *variable tree* (in short, a *vtree*) for the variable set Y is a rooted, full, ordered, binary tree T whose leaves correspond bijectively to Y; for simplicity, we identify each leaf in T with the variable in Y it corresponds to.

For every internal node v of the vtree T, we let v_l and v_r denote respectively the left and right child of v. Moreover, we denote by T_v the subtree of T rooted at v. We also let $Y_v \subseteq Y$ denote (the variables corresponding to) the leaves of T_v.

Let C be a DNNF over variables X, and let T be a vtree for the variable set Y. Let g be an $\wedge$-gate in C having wires from gates h and h'. We say that g *respects* the node v of T if $\mathsf{var}(C_h) \subseteq Y_{v_l}$ and $\mathsf{var}(C_{h'}) \subseteq Y_{v_r}$. We say that C *respects* the vtree T if each $\wedge$-gate in C respects some node in T. A DNNF C is called *structured* if it respects some vtree.

We introduce the notion of rectangle, the combinatorial core of communication complexity.

Rectangles and Covers. Let X be a finite set of variables. A partition of X is a sequence of pairwise disjoint subsets (blocks) of X whose union is X. A partition (X_1, X_2) of X into two blocks is called *balanced* if $|X|/3 \leq \min(|X_1|, |X_2|)$; clearly, this is equivalent to $\max(|X_1|, |X_2|) \leq 2|X|/3$.

Let (X_1, X_2) be a partition of X. For $b_1 \colon X_1 \to \{0,1\}$ and $b_2 \colon X_2 \to \{0,1\}$, we let $b_1 \cup b_2 \colon X_1 \cup X_2 \to \{0,1\}$ denote the assignment whose restriction to X_i equals b_i for $i = 1, 2$. Also, for $B_1 \subseteq \{0,1\}^{X_1}$ and $B_2 \subseteq \{0,1\}^{X_2}$, we let $B_1 \times B_2 = \{b_1 \cup b_2 \mid b_1 \in B_1, b_2 \in B_2\}$.

A *(combinatorial) rectangle* over X is a function $r \colon \{0,1\}^X \to \{0,1\}$ such that there exist an *underlying* partition (X_1, X_2) of X and functions $r^i \colon \{0,1\}^{X_i} \to \{0,1\}$ for $i = 1, 2$ such that $\mathsf{sat}(r) = \mathsf{sat}(r^1) \times \mathsf{sat}(r^2)$. A rectangle is called *balanced* if its underlying partition is balanced.

We also call a subset R of $\{0,1\}^X$ a rectangle over X, with underlying partition (X_1, X_2), if there exists a rectangle $r\colon \{0,1\}^X \to \{0,1\}$, with underlying partition (X_1, X_2), such that $R = \mathsf{sat}(r)$.

Let $f\colon \{0,1\}^X \to \{0,1\}$ be a function. A finite set $\{r_i\}$ of rectangles over X is called a *rectangle cover* of f if

$$\mathsf{sat}(f) = \bigcup_i \mathsf{sat}(r_i); \tag{6.1}$$

the rectangle cover is called *disjoint* if the union in (6.1) is disjoint. A rectangle cover is called *balanced* if each rectangle in the cover is balanced.

Note that, if f has a rectangle cover as in (6.1), then$f \equiv \bigvee_i \left(C_i^1 \wedge C_i^2\right)$, where C_i^1 (respectively, C_i^2) is an NNF over the first (respectively, second) block of the partition underlying the ith rectangle in the cover; the outermost $\vee$-gate is deterministic if the cover is disjoint, and the ith $\wedge$-gate displayed is decomposable (by the partition of the ith rectangle).

6.2. Knowledge Compilation Meets Communication Complexity

In this section we show how to construct, given a (deterministic) DNNF C, a (disjoint) rectangle cover of size at most $|C|$ for f_C (Theorem 6.7), thus establishing a fundamental connection between knowledge compilation and communication complexity.

The construction is based on basic but crucial combinatorial properties of (deterministic) DNNFs, notably on the notion of certificate for a DNNF, and an operation eliminating gates in DNNFs, which we now define and study.

Let C be a DNNF on variables X. A *certificate* of C is a DNNF T on variables X such that: T contains the output gate of C; if T contains an $\wedge$-gate v of C, then T also contains every gate of C having an output wire to v; if T contains an $\vee$-gate v of C, then T also contains exactly one gate of C having an output wire to v. The output gate of T coincides with the output gate of C, and the gates of T inherit their labels and wires from C.

We let $\mathsf{cert}(C)$ denote the set of certificates of C. It is readily verified that

$$\mathsf{sat}(C) = \bigcup_{T \in \mathsf{cert}(C)} \mathsf{sat}(T). \tag{6.2}$$

The following fact is an easy consequence of decomposability (and constant freeness).

Fact 6.1. *Let C be a DNNF and let $T \in \mathsf{cert}(C)$. The graph underlying T is a binary tree. Moreover, no two leaves of T are labeled by the same variable.*

For a gate g of a DNNF C, we let $\mathsf{cert}(C, g)$ denote the set of certificates of C containing the gate g and let

$$\mathsf{sat}(C, g) = \bigcup_{T \in \mathsf{cert}(C,g)} \mathsf{sat}(T); \tag{6.3}$$

in words, $\mathsf{sat}(C, g)$ contains those satisfying assignments of C that satisfy the subcircuit rooted at gate g.

The crucial combinatorial property of DNNFs is that $\mathsf{sat}(C, g)$ is a rectangle separating the variables in the subcircuit of C rooted at g. Formally,

Theorem 6.2. *Let C be a DNNF on variables X and let g be a gate of C. Then $\mathsf{sat}(C, g)$ is a rectangle over X with underlying partition $(\mathsf{var}(C_g), X \setminus \mathsf{var}(C_g))$.*

In view of proving Theorem 6.2, we prepare the following.

Lemma 6.3. *Let g be a gate of a DNNF C and let $T \in \mathsf{cert}(C, g)$. Then*

$$\mathsf{var}(T) \setminus \mathsf{var}(T_g) \subseteq \mathsf{var}(C) \setminus \mathsf{var}(C_g).$$

Proof of Lemma 6.3. Otherwise, let x be a variable contained in both $\mathsf{var}(T) \setminus \mathsf{var}(T_g)$ and $\mathsf{var}(C_g)$. Then there exists a certificate $T' \in \mathsf{cert}(C, g)$ such that $x \in \mathsf{var}(T'_g)$. By Fact 6.1, T and T' are trees. By replacing T_g in T by T'_g, we obtain a certificate T of C where x occurs twice, contradicting Fact 6.1. □

Proof of Theorem 6.2. Let $Y = \mathsf{var}(C_g)$ and $Y' = X \setminus Y$. Let b and b' be in $\mathsf{sat}(C, g)$. It is sufficient to show that $b|_Y \cup b'|_{Y'}$ is in $\mathsf{sat}(C, g)$, where $b|_Y$, denotes the restriction of b to Y and $b'|_{Y'}$ denotes the restriction of b' to Y'.

By (6.3), there exist certificates T and T' in $\mathsf{cert}(C, g)$ such that $b \in \mathsf{sat}(T)$ and $b' \in \mathsf{sat}(T')$. Then b satisfies all literals in T_g. Since $\mathsf{var}(T_g) \subseteq Y$, it follows that $b|_Y$ satisfies all literals in T_g. Similarly, b' satisfies all literals in $T' \setminus T'_g$; and, by Lemma 6.3, it holds that $\mathsf{var}(T') \setminus \mathsf{var}(T'_g) \subseteq Y'$. Hence $b'|_{Y'}$ satisfies all literals in $T' \setminus T'_g$.

By Fact 6.1, T and T' are trees. By replacing T'_g in T' by T_g, we obtain a certificate S of C containing g, that is, $S \in \mathsf{cert}(C, g)$. By the above observations, $b|_Y \cup b'|_{Y'}$ satisfies all literals in S, that is, $b|_Y \cup b'|_{Y'}$ is in $\mathsf{sat}(S)$. It follows by (6.3) that $b|_Y \cup b'|_{Y'}$ is in $\mathsf{sat}(C, g)$. □

As $\mathsf{cert}(C) = \bigcup_{g \in C} \mathsf{cert}(C, g)$, it trivially follows by (6.2) and (6.3) that

$$\mathsf{sat}(C) = \bigcup_{g \in C} \mathsf{sat}(C, g). \tag{6.4}$$

In words, C is covered by the rectangles induced by its gates (recall Theorem 6.2). However, in view of reusing known lower bounds on the size of rectangle covers (see Section 6.3), we need to find a subset of gates of C generating a *balanced* rectangle cover for C.

To this aim, we first introduce and study an operation on DNNFs that boils down to relabelling a non-input gate by 0 and propagating the information in the circuit.

Let C be a DNNF on variables X. We define an operation (0-*propagation*) that, given a DNNF C with some gates labelled with 0, returns either a single gate labelled with a 0 or a DNNF where no gates are labelled with 0. The operation iterates the following step until all 0s disappear (or the DNNF reduces to a single gate labelled 0). Let g be a gate in C labelled with 0. Then: delete all input wires of g; delete all output wires of g to

$\vee$-gates; relabel all $\wedge$-gates wired by g and all fanin 0 $\vee$-gates by 0; delete all gates with no directed paths to the output gate.

Now we define the DNNF on variables X obtained by *eliminating the non-input gate g in C*, denoted by $C - g$, as the result of relabelling g by 0 and performing 0-propagation.

The impact of passing from C to $C - g$ is dropping all certificates containing g in (6.2), as formalized by the following proposition.

Proposition 6.4. *Let C be a DNNF and let g be a non-input gate of C. Then*

$$\mathsf{sat}(C - g) = \bigcup_{T \in \mathsf{cert}(C) \setminus \mathsf{cert}(C,g)} \mathsf{sat}(T).$$

Proof. Let first $b \in \mathsf{sat}(C - g)$. Then there must be a certificate T in $C - g$ with $b \in \mathsf{sat}(T)$. Since g is not in T, it follows that $T \in \mathsf{cert}(C) \setminus \mathsf{cert}(C, g)$. Thus $b \in \bigcup_{T \in \mathsf{cert}(C) \setminus \mathsf{cert}(C,g)} \mathsf{sat}(T)$ and thus $\mathsf{sat}(C - g) \subseteq \bigcup_{T \in \mathsf{cert}(C) \setminus \mathsf{cert}(C,g)} \mathsf{sat}(T)$.

For the other direction, assume that $b \in \bigcup_{T \in \mathsf{cert}(C) \setminus \mathsf{cert}(C,g)} \mathsf{sat}(T)$. Then there is a certificate $T \in \mathsf{cert}(C) \setminus \mathsf{cert}(C, g)$ with $b \in \mathsf{sat}(T)$. Since g is not in T and 0-propagation does not affect any certificates that do not contain g, we get that $T \in \mathsf{cert}(C - g)$. It follows that $b \in \mathsf{sat}(C - g)$ and thus $\bigcup_{T \in \mathsf{cert}(C) \setminus \mathsf{cert}(C,g)} \mathsf{sat}(T) \subseteq \mathsf{sat}(C - g)$. □

The following lemma states a property of gate elimination crucial to our construction: in passing from C to $C - g$ we *only* forget satisfying assignments in the rectangle $\mathsf{sat}(C, g)$.

Lemma 6.5. *Let C be a DNNF and let g be a non-input gate of C. Then $C - g$ is a DNNF and*

$$\mathsf{sat}(C) \setminus \mathsf{sat}(C, g) \subseteq \mathsf{sat}(C - g) \subseteq \mathsf{sat}(C).$$

Proof. Note that gate elimination preserves decomposability. The inclusions follow directly from Proposition 6.4, recalling (6.2) and (6.3). □

In general, an assignment can satisfy more than one certificate. In this case, the left inclusion in Lemma 6.5 is strict. For instance, let D be a DNNF and let $C = D \vee D$. Let g be the output gate of one copy of D in C. Then $\mathsf{sat}(C) = \mathsf{sat}(D)$ and $\mathsf{sat}(C, g) = \mathsf{sat}(D)$, so that $\mathsf{sat}(C) \setminus \mathsf{sat}(C, g) = \emptyset$, but $\mathsf{sat}(C - g) = \mathsf{sat}(D)$.

By contrast, the left inclusion in Lemma 6.5 becomes an equality in the deterministic case; in other words, eliminating a gate g in a deterministic DNNF C removes *exactly* the assignments in the rectangle $\mathsf{sat}(C, g)$. Formally,

Lemma 6.6. *Let C be a deterministic DNNF and let g be a non-input gate of C. Then $C - g$ is a deterministic DNNF and*

$$\mathsf{sat}(C) \setminus \mathsf{sat}(C, g) = \mathsf{sat}(C - g).$$

Proof. We show that gate elimination preserves determinism. Assume that $C - g$ contains a nondeterministic $\vee$-gate h, wired by gates k and k' such that $b \in \mathsf{sat}((C - g)_k)$ and $b \in \mathsf{sat}((C - g)_{k'})$. It follows by Proposition 6.4 that there exist certificates T and T' in $\mathsf{cert}(C) \setminus \mathsf{cert}(C, g)$ such that $b \in \mathsf{sat}(T_k)$ and $b \in \mathsf{sat}(T'_k)$. Then $b \in \mathsf{sat}(C_k)$ and $b \in \mathsf{sat}(C_{k'})$, that is, h is nondeterministic in C, a contradiction.

For the equality, by Lemma 6.5 it suffices to prove that $\mathsf{sat}(C - g)$ is contained in $\mathsf{sat}(C) \setminus \mathsf{sat}(C, g)$. Assume $b \in \mathsf{sat}(C - g)$ so that, by Proposition 6.4, it holds that $b \in \mathsf{sat}(T')$ for some $T' \in \mathsf{cert}(C) \setminus \mathsf{cert}(C, g)$. In particular, $b \in \mathsf{sat}(C)$. It suffices to show that $b \notin \mathsf{sat}(C, g)$.

Otherwise, by (6.3), $b \in \mathsf{sat}(T)$ for some $T \in \mathsf{cert}(C, g)$. Since $T' \notin \mathsf{cert}(C, g)$, we have $T \neq T'$. It follows that there exist two distinct gates k and k' in C, wiring an $\vee$-gate h in C, such that T contains k and T' contains k'. Then $b \in \mathsf{sat}(T_k)$ and $b \in \mathsf{sat}(T_{k'})$, so that $b \in \mathsf{sat}(C_k)$ and $b \in \mathsf{sat}(C_{k'})$. Again, h is nondeterministic in C, a contradiction. □

It follows from Lemma 6.5 that the process of iteratively eliminating gates in a DNNF (until it becomes unsatisfiable) yields a rectangle cover; moreover, by Lemma 6.6, the rectangle cover is disjoint if the DNNF is deterministic.

We strengthen the above remark by proving that a suitable elimination sequence in a (deterministic) DNNF C yields not just a (disjoint) rectangle cover of C, but indeed a *balanced* one, a crucial feature for the intended application (Section 6.3).

Theorem 6.7. *Let C be a (deterministic) DNNF computing a function f. Then f has a balanced (disjoint) rectangle cover of size at most $|C|$.*

Proof. Let $C = C^0$ be a (deterministic) DNNF over variables $X = \mathsf{var}(C)$ computing f. For $i = 0, 1, \ldots$, we find a suitable gate $g_i \in C^i$ and construct the (deterministic) DNNF $C^{i+1} = C^i - g_i$ by eliminating g_i in C^i, until we hit $l \leq |C|$ such that $C^l \equiv 0$. Along the way, we construct a set

$$\{R_i \mid i = 0, \ldots, l-1\} \tag{6.5}$$

that, we claim, is the desired rectangle cover of f.

For $i = 0, 1, \ldots$, we choose the gate g_i as follows. We distinguish two cases. If $2|X|/3 < |\mathsf{var}(C^i)|$ then, by a descent from the output gate of C^i, we find a gate $g_i \in C^i$ such that $|X|/3 \leq |\mathsf{var}(C^i_{g_i})| \leq 2|X|/3$. By Theorem 6.2 we have that $\mathsf{sat}(C^i, g_i)$ is a rectangle over X with underlying partition $(\mathsf{var}(C^i_{g_i}), X \setminus \mathsf{var}(C^i_{g_i}))$. Then $R_i = \mathsf{sat}(C^i, g_i)$ is a balanced rectangle over X.

If $|\mathsf{var}(C^i)| < 2|X|/3$ then we let g_i be the root of C^i. We obtain the desired rectangle as follows. Let $\mathsf{var}(C^i) \subseteq X' \subseteq X$ be obtained by adding to $\mathsf{var}(C^i)$ enough variables from $X \setminus \mathsf{var}(C^i)$ so that $(X', X \setminus X')$ is a balanced partition of X. We put

$$R_i = (\mathsf{sat}(C^i) \times \{0,1\}^{X' \setminus \mathsf{var}(C^i)}) \times \{0,1\}^{X \setminus X'},$$

where we view C^i as a DNNF over $\mathsf{var}(C^i)$. Then R_i is trivially a balanced rectangle over X.

It follows from the above construction and Lemma 6.5 that the set in (6.5) is a balanced rectangle cover of C. Moreover, if C is deterministic, then such rectangle cover is disjoint, because $\mathsf{sat}(C^i, g_i) \cap \mathsf{sat}(C^{i+1}) = \emptyset$ by Lemma 6.6. □

	NNF	DNNF	DNF	CNF	PI	IP
DNNF	$\not\leq^{**}$	$\leq$	$\leq$	$\not\leq^{**}$	$\not\leq^{**}$	$\leq$
d-DNNF	$\not\leq^{**}$	$\not\leq^{*}$	?	$\not\leq^{**}$	$\not\leq^{**}$	?

Table 6.1.: Relative succinctness of knowledge compilation languages, taking into account the results of Section 6.3.1 (*) and Section 6.3.2 (**).

6.3. Separating Knowledge Representation Languages via Communication Complexity

In this section, we combine the connection between (deterministic) DNNFs and (disjoint) rectangle covers established in Section 6.2 with deep combinatorial lower bounds on the size of (disjoint) rectangle covers from the communication complexity literature to obtain exponential separations of DNNFs from deterministic DNNFs (Section 6.3.1) and of prime implicates (PI) from DNNFs (Section 6.3.2).

As illustrated in Table 6.1, these results allow us to answer several questions regarding the relative succinctness of languages left open in the "knowledge compilation map" (cf. Table 3 of [DM02]).

6.3.1. DNNFs Versus Deterministic DNNFs

We first prove an exponential separation of DNNFs from deterministic DNNFs. The two classes are separated by a function introduced and studied by Sauerhoff [Sau03].

Let $g_n\colon \{0,1\}^n \to \{0,1\}$ be the function evaluating to 1 if and only if the sum of its inputs is divisible by 3. The *Sauerhoff function* $S_n\colon \{0,1\}^{n^2} \to \{0,1\}$ is defined on the $n \times n$ matrix $X = (x_{ij})_{1\leq i,j\leq n}$ of variables by

$$S_n(X) = R_n(X) \vee C_n(X) \tag{6.6}$$

where $R_n, C_n\colon \{0,1\}^{n^2} \to \{0,1\}$ are defined by

$$R_n(X) = \bigoplus_{i=1}^{n} g_n(x_{i1}, x_{i2}, \ldots, x_{in})$$

and $C_n(X) = R_n(X^\top)$, where $X^\top$ denotes the transpose of X, and $\oplus$ denotes addition modulo 2.

The Sauerhoff function has polynomial DNNF size.

Proposition 6.8. *S_n in (6.6) has DNNF size $O(n^2)$.*

Proof (Sketch). The functions R_n and C_n have OBDDs of size $O(n^2)$, ordering the variables by rows and columns, respectively; their disjunction has size $O(n^2)$. □

We use a highly nontrivial exponential lower bound on the size of balanced disjoint rectangle covers for S_n [Sau03, Theorem 4.10].

Theorem 6.9 (Sauerhoff)**.** *Any balanced disjoint rectangle cover of the Sauerhoff function S_n in (6.6) has size $2^{\Omega(n)}$.*

We remark that Sauerhoff actually proves the above lower bound only for rectangles whose underlying partitions have blocks of the same size ± 1, but a careful inspection of the proof reveals that the same argument can be lifted to our more relaxed notion of balancedness.

By putting together Theorem 6.7 and Theorem 6.9, we get the following lower bound, which, in combination with Proposition 6.8, yields an explicit, unconditional, exponential separation of DNNFs from deterministic DNNFs:

Theorem 6.10. *S_n in (6.6) has deterministic DNNF size $2^{\Omega(n)}$.*

Proof. Let C be a deterministic DNNF computing S_n. By Theorem 6.7, the size of a balanced disjoint rectangle cover of S_n bounds below the size of C; but the size of this cover is $2^{\Omega(n)}$ by Theorem 6.9. □

6.3.2. Prime Implicates Versus DNNFs

Next, we show a (strongly) exponential separation of prime implicates (PIs) from DNNFs. In recent (unpublished) work [43], we established this separation by means of an involved combinatorial proof; here, we obtain the same result by leveraging a lower bound on the multi-partition communication complexity of a function studied by Jukna and Schnitger [JS02], which is defined as follows.

For $n \geq 2$, let K_n be the set of all 2-element subsets (edges) of $\{1, \ldots, n\}$. We view every subset of K_n as the edge set of a graph G whose vertex set is $\{1, \ldots, n\}$. We identify edges in K_n with Boolean variables, so that the graph $G \subseteq K_n$ is encoded by the $\{0,1\}$-assignment of K_n mapping a variable (edge) to 1 if and only if it is in the edge set of G.

A triangle T on n vertices is a graph with vertices $\{1, \ldots, n\}$ and an edge set $\{\{i,j\},\{i,k\},\{j,k\}\}$, where $|\{i,j,k\}| = 3$; it corresponds to the assignment of K_n mapping $\{i,j\}$, $\{i,k\}$, $\{j,k\}$ to 1 and the other edges to 0.

We let $\mathcal{T}_n$ be the set of all triangles on n vertices. For a set $A \subseteq \mathcal{T}_n$, we let

$$JS_n^A \colon \{0,1\}^{K_n} \to \{0,1\} \tag{6.7}$$

denote the function accepting exactly those graphs over $\{1, \ldots, n\}$ that avoid all triangles in A (the edge set of no triangle in A is contained in the edge set of the input graph).

Jukna and Schnitger [JS02, Theorem 3.1] show an exponential lower bound on the size of balanced rectangle covers for functions as in (6.7).

Theorem 6.11 (Jukna and Schnitger)**.** *For every n there exists $A_n \subseteq \mathcal{T}_n$ of size $O(n^2)$ such that any balanced rectangle cover of $JS_n^{A_n}$ in (6.7) has size $2^{\Omega(n^2)}$.*

The *Jukna-Schnitger function* $JS_n \colon \{0,1\}^{K_n} \to \{0,1\}$ is defined by

$$JS_n = JS_n^{A_n} \tag{6.8}$$

where A_n is chosen by Theorem 6.11 ($n \geq 2$).

It is readily verified that the Jukna-Schnitger function has polynomial PI size. Recall that a CNF F is in *prime implicate* (*PI*) form if every clause entailed by F is already entailed by a clause of F, and no clause of F entails another clause of F.

Proposition 6.12. *JS_n in (6.8) has PI size $O(n^2)$.*

Proof (Sketch). Let $JS_n = JS_n^{A_n}$. Take the CNF F_n stating that every triangle in A_n has an edge that is not in the input graph; it computes JS_n and it is in PI. Also, F_n has size $O(n^2)$, since $|A_n| = O(n^2)$ by Theorem 6.11. □

By combining Theorem 6.7 and Theorem 6.11, we obtain the following lower bound.

Theorem 6.13. *JS_n in (6.8) has DNNF size $2^{\Omega(n^2)}$.*

Proof. Let C be a DNNF computing JS_n. The size of a balanced rectangle cover of JS_n is at most $|C|$ by Theorem 6.7 and at least $2^{\Omega(n^2)}$ by Theorem 6.11. □

Jointly, Proposition 6.12 and Theorem 6.13 yield an unconditional, strongly exponential separation of PIs from DNNFs. As already observed in [43], since PI $\subseteq$ CNF $\subseteq$ NNF and d-DNNF $\subseteq$ DNNF, the remaining separations in Table 6.1 marked with ** follow from this result.

6.4. Structured Knowledge Representation Languages and Communication Complexity

The lower bound techniques for structured DNNFs introduced by Pipatsrisawat and Darwiche [PD10] have a natural interpretation in terms of communication complexity. Their main result can be paraphrased thus:

Theorem 6.14 (Pipatsrisawat and Darwiche)**.** *Let D be a (deterministic) structured DNNF on variables X computing a function f and respecting a vtree T. For every node $v \in T$, f has a (disjoint) rectangle cover of size at most $|D|$ where each rectangle has underlying partition $(X_v, X \setminus X_v)$.*

Proof (Sketch). Let v be a node in T. We can find a gate g of D such that, for every certificate C of D containing g, it holds that $\mathsf{var}(C_g) \subseteq X_v$ and $\mathsf{var}(C \setminus C_g) \subseteq X \setminus X_v$. We can show as in Theorem 6.2 that $\mathsf{sat}(D, g)$ is thus a rectangle with underlying partition $(X_v, X \setminus X_v)$. We then apply a similar elimination process as in the proof of Theorem 6.7. □

In contrast to Theorem 6.7, the above statement speaks about rectangle covers whose rectangles share the same underlying partition. Such covers are closely related to a measure known as the best-partition communication complexity [LS81], and Theorem 6.14 allows us to transfer lower bounds on the best-partition communication complexity and prove a conjecture by Pipatsrisawat and Darwiche [PD10].

Let $X_n = \{x_1, \ldots, x_n\}$ and $Y_n = \{y_1, \ldots, y_n\}$. Let T be a vtree for $X_n \cup Y_n$ where the subtree rooted at the left (respectively, right) child of the root is a right-linear vtree for X_n (respectively, Y_n). Pipatsrisawat and Darwiche [PD10] conjecture that any deterministic DNNF computing

$$f_n = (x_1 \wedge y_1) \vee \cdots \vee (x_n \wedge y_n) \tag{6.9}$$

and respecting T has size exponential in n. We appeal to a nice piece of communication complexity theory to prove the following statement (thus confirming the conjecture).

Proposition 6.15. *Let T be any vtree for $X_n \cup Y_n$ containing a vtree for X_n as a subtree. Then any deterministic DNNF computing f_n in (6.9) and respecting T has size at least $2^n - 1$.*

Let $f\colon \{0,1\}^Z \to \{0,1\}$ be a function, and let (Z_1, Z_2) be a partition of X where $|Z_1| = |Z_2| = n$. The *communication matrix* of f relative to (Z_1, Z_2), denoted by $M(f, Z_1, Z_2)$ is a (Boolean) matrix whose rows and columns are indexed by assignments of Z_1 and Z_2, respectively, and whose (b_1, b_2)th entry equals $f(b_1 \cup b_2)$.[3]

A basic fact in communication complexity is that the rank of the communication matrix is a lower bound on the size of disjoint rectangle covers of a function [Juk12, Section 4.1].

Theorem 6.16. *Let (Z_1, Z_2) be a partition of the variables of a function f, where $|Z_1| = |Z_2| = n$. Then every disjoint rectangle cover of f into rectangles with underlying partition (Z_1, Z_2) has size at least* $\operatorname{rank}(M(f, Z_1, Z_2))$.

The complement of the function f_n in (6.9), called the *disjointness function*,

$$d_n = \neg f_n = (\neg x_1 \vee \neg y_1) \wedge \cdots \wedge (\neg x_n \vee \neg y_n), \tag{6.10}$$

is a well studied object in communication complexity; we denote by $D_n = M(d_n, X_n, Y_n)$ its communication matrix. The following fact is folklore [Juk12, Exercise 7.1].

Proposition 6.17. $\operatorname{rank}(D_n) = 2^n$.

Proof (Sketch). Note that

$$D_n = \begin{pmatrix} D_{n-1} & D_{n-1} \\ D_{n-1} & 0 \end{pmatrix}$$

and that $\operatorname{rank}(D_1) = 2$. The statement follows by induction on n since $\operatorname{rank}(D_n) = \operatorname{rank}(D_{n-1}) \cdot \operatorname{rank}(D_1)$. □

Proof of Proposition 6.15. Let C be any deterministic DNNF computing f_n in (6.9) and respecting a vtree T as in the hypothesis. By Theorem 6.14, f_n has a disjoint rectangle cover of size at most $|C|$ where each rectangle has underlying partition (X_n, Y_n). Let $E_n = M(f_n, X_n, Y_n)$. By Theorem 6.16, $|C| \geq \operatorname{rank}(E_n)$. Since $D_n = 1 - E_n$ by (6.10), we have $2^n = \operatorname{rank}(D_n) = \operatorname{rank}(1 - E_n) \leq \operatorname{rank}(1) + \operatorname{rank}(E_n) = 1 + \operatorname{rank}(E_n)$ by Proposition 6.17 and basic linear algebra, hence $\operatorname{rank}(E_n) \geq 2^n - 1$. We conclude that $|C| \geq 2^n - 1$. □

[3] We regard communication matrices as matrices over the reals.

We conclude by noting that the same general strategy used for obtaining the lower bounds in Section 6.3 works for structured DNNFs. For instance, the exponential lower bound on the structured DNNF size of the circular bit shift (CBS) function [PD10], follows directly by Theorem 6.14 and known exponential lower bounds on the size of rectangle covers for CBS in the best-partition model [KN97, Chapter 7.2].

6.5. Conclusion

We established a connection between the DNNF size and the multi-partition communication complexity of Boolean functions. This connection allowed us to translate lower bounds from communication complexity into lower bounds on the (deterministic) DNNF size and prove exponential separations of DNNFs from d-DNNFs and of PIs from DNNFs.

We are confident that the applicability of our approach goes beyond the specific lower bound results proved here. In particular, we hope that it can help resolve a few questions from the "knowledge compilation map" that remain open [DM02].

7. Lower Bounds for Approximate Compilation

In this chapter, we give most of the details of the results on approximate knowledge compilation from [15] which were discussed in Section 2.5. The proofs show how the framework of [27] can be used to show results beyond its initial scope in exact compilation.

We start with some preliminaries in Section 7.1 before introducing the notion of weak approximation and giving our lower bound for it in Section 7.2. Afterwards, we introduce and discuss strong approximations next in Section 7.3 and show that weak and strong approximations differ in Section 7.4. To keep this chapter relatively light, most of the details of Section 7.2 are not presented, since they are essentially only a minimally changed version of a similar proof in [KSW99]; the interested reader finds them in the arXiv version of [15].

7.1. Preliminaries

We describe some conventions of notation for Boolean algebra. In our framework, a Boolean variable takes value 0 ($false$) or 1 ($true$), we see it as a variable over $\mathbb{F}_2$, the field with two elements. Assignments of n Boolean variables are vectors from $\mathbb{F}_2^n$ and operations on vectors and matrices are considered in this field. We use the notation $\mathbf{0}^n$ to denote the 0-vector from $\mathbb{F}_2^n$. For clarity we also use the operators $\neg$, $\vee$ and $\wedge$ for negation, disjunction and conjunction in $\mathbb{F}_2$. The conjunction of Boolean variables and the product in $\mathbb{F}_2$ are equivalent and used interchangeably. Single variables are written in plain style "x" while assignments of $n > 1$ variables use bold style "$\mathbf{x}$". A Boolean function on n variables is a mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2$ and its models are given by $f^{-1}(1)$. Given a set of assignments S, we sometimes denote $\mathbf{1}_S$ the Boolean function whose set of models is exactly S. We write $f \leq g$ when $f^{-1}(1) \subseteq g^{-1}(1)$, which corresponds to logical entailment. A distribution on truth assignments is a probabilistic distribution $\mathcal{D}$ on $\mathbb{F}_2^n$. We write $\Pr_{\mathbf{x}\sim\mathcal{D}}[\cdot]$ to denote the probability measure when sampling an assignment $\mathbf{x}$ according to $\mathcal{D}$. For clarity, the uniform distribution on $\mathbb{F}_2^n$ is denoted $\mathcal{U}$ (regardless of n), $\mathbf{x} \sim \mathcal{U}$ means that any assignment is sampled with probability $1/2^n$.

Deterministic decomposable NNF. Let X be a finite set of Boolean variables. A circuit in *negation normal form* (NNF) over X is a single output Boolean circuit whose inputs gates are labeled with Boolean variables x from X and their negations $\neg x$ and whose internal gates are fanin-2 $\wedge$- and $\vee$-gates. The *size* of a circuit is the number of its gates. A circuit over X is said to *accept* a truth assignment $\mathbf{x}$ of the variables if it outputs 1

(*true*) when its inputs are set as in $\mathbf{x}$. In this case $\mathbf{x}$ is a *model* of the function represented by the circuit. An NNF is *decomposable* if, for any $\wedge$-gate g, the two sub-circuits rooted at g share no input variable, i.e., if x or $\neg x$ is an input gate of the circuit rooted at the left input of g, then neither x nor $\neg x$ is an input gate of the subcircuit rooted at the right input, and vice versa. An NNF is *deterministic* if, for any $\vee$-gate g, the sets of assignments accepted by the two subcircuits rooted at the children of g are disjoint. A decomposable NNF is called a DNNF; if in addition it is deterministic, then it is called a d-DNNF.

Rectangle covers. Let X be a finite set of Boolean variables. A *combinatorial rectangle* over X (more succinctly a *rectangle*) is a Boolean function r defined as the conjunction of two Boolean functions ρ_1 and ρ_2 over disjoints variables of X. That is, there is a partition (X_1, X_2) of X such that ρ_1 and ρ_2 are defined over X_1 and X_2, respectively, and $r = \rho_1 \wedge \rho_2$. We call (X_1, X_2) the *partition* of r. The rectangle is *balanced* if $|X|/3 \leq |X_1| \leq 2|X|/3$ (the same bounds hold for $|X_2|$). A *rectangle cover* of a Boolean function f is any disjunction of rectangles over X (possibly for different partitions of X) equivalent to f, i.e., $f = \bigvee_{i=1}^{K} r_i$ where the r_i are rectangles. The *size* of a cover is the number K of its rectangles. A rectangle cover is called *balanced* if its rectangles are balanced and it is said *disjoint* if no two rectangles share a model. Note that any function f has at least one balanced disjoint rectangle cover, because it can be written as a DNF in which every term contains all variables. There is a tight link between the smallest size of a balanced disjoint rectangle cover of a function and the size of any equivalent d-DNNF.

Theorem 7.1. [27] *Let D be a d-DNNF encoding a function f. Then f has a balanced disjoint rectangle cover of size at most the size of D.*

Theorem 7.1 implies that, to show a lower bound on the size of any d-DNNF encoding f, it is sufficient to find a lower bound on the size of any balanced disjoint rectangle cover of f.

7.2. Large d-DNNFs for Weak Approximations

In this section, we start by considering the notion of approximation that has been studied for different forms of branching programs before, see e.g. [KSW99, BSW02]. To differentiate it from other notions, we give it the name *weak approximation*.

Definition 7.2 (Weak approximation)**.** Let $\mathcal{D}$ be a distribution on the truth assignments to X and $\varepsilon > 0$. We say that $\tilde{f}$ is a *weak ε-approximation* of f (or weakly ε-approximates f) with respect to $\mathcal{D}$ if

$$\Pr_{\mathbf{x} \sim \mathcal{D}} \left[f(\mathbf{x}) \neq \tilde{f}(\mathbf{x}) \right] \leq \varepsilon.$$

When $\mathcal{D}$ is the uniform distribution $\mathcal{U}$, then the condition of weak ε-approximability is equivalent to $|\{\mathbf{x} : f(\mathbf{x}) \neq \tilde{f}(\mathbf{x})\}| \leq \varepsilon 2^n$.

Note that weak ε-approximation is only useful when $\varepsilon < 1/2$. This is because every function has a trivial $(1/2)$-approximation: if $\Pr_{\mathbf{x}\sim\mathcal{D}}[f(\mathbf{x}) = 1] > 1/2$, then the constant 1-function is a $(1/2)$-approximation, otherwise this is the case for the constant 0-function. Note that it might be hard to decide which case is true, but in any case we know that the approximation ratio of one of the constants is good.

Bollig *et al.* [BSW02] used a *discrepancy* argument to show that there are classes of functions such that any ε-approximation w.r.t. $\mathcal{U}$ requires exponential OBDD size. We lift their techniques to d-DNNF showing that the same functions are also hard for d-DNNF.

Theorem 7.3. *Let $0 \leq \varepsilon < 1/2$, there is a class of Boolean functions $\mathcal{C}$ such that, for any $f \in \mathcal{C}$ on n variables, any d-DNNF encoding a weak ε-approximation of f w.r.t. $\mathcal{U}$ has size $2^{\Omega(n)}$.*

Since d-DNNFs are strictly more succinct than OBDDs [DM02], Theorem 7.3 is a generalization of the result on OBDDs in [BSW02]. However, since the proof is quite long and almost identical, differing near the end only, we defer the technical details to the arXiv version of [15]. We here only introduce the notion of discrepancy that is central to the proof and will be useful later.

The discrepancy method. We want to use Theorem 7.1 to bound the size of a d-DNNF encoding $\tilde{f}$ a weak ε-approximation of f w.r.t. some distribution. To this end we study disjoint balanced rectangle covers of $\tilde{f}$. Let r be a rectangle from such a cover. r can make *false positives* on f, i.e., have models that are not models of f. Similarly, *true positives* are models shared by r and f. The *discrepancy* $\mathrm{Disc}(f, r)$ of f on r is the difference between the number of false positives and true positives, normalized by the total number of assignments: $\mathrm{Disc}(f, r) := \frac{1}{2^n}\big||r^{-1}(1) \cap f^{-1}(1)| - |r^{-1}(1) \cap f^{-1}(0)|\big|$. A small discrepancy indicates that r has few models or that it makes roughly as many false positives as true positives on f. Discrepancy bounds have been used before to prove results in distributional communication complexity [KN97, Chapter 3.5]. Here we show that when there is an upper bound on $\mathrm{Disc}(f, r)$ for any rectangle r from a cover of $\tilde{f}$, one can obtain a lower bound on the size of the cover of $\tilde{f}$.

Lemma 7.4. *Let f be a Boolean function on n variables and let $\tilde{f}$ be a weak ε-approximation of f w.r.t. $\mathcal{U}$. Let $\tilde{f} = \bigvee_{k=1}^{K} r_k$ be a disjoint balanced rectangle cover of $\tilde{f}$ and assume that there is an integer $\Delta > 0$ such that $\mathrm{Disc}(f, r_k) \leq \Delta/2^n$ for all r_k. Then $K \geq (|f^{-1}(1)| - \varepsilon 2^n)/\Delta$.*

Proof. We have $|f \neq \tilde{f}| = |\{\mathbf{x} : f(\mathbf{x}) \neq \tilde{f}(\mathbf{x})\}|$

$$
\begin{aligned}
&= |f^{-1}(1) \cap \tilde{f}^{-1}(0)| + |f^{-1}(0) \cap \tilde{f}^{-1}(1)| \\
&= \Big|f^{-1}(1) \cap \bigcap\nolimits_{k=1}^{K} r_k^{-1}(0)\Big| + \Big|f^{-1}(0) \cap \bigcup\nolimits_{k=1}^{K} r_k^{-1}(1)\Big| \\
&= |f^{-1}(1)| - \sum_{k=1}^{K}(|r_k^{-1}(1) \cap f^{-1}(1)| - |r_k^{-1}(1) \cap f^{-1}(0)|) \\
&\geq |f^{-1}(1)| - 2^n \sum\nolimits_{k=1}^{K} \mathrm{Disc}(f, r_k) \geq |f^{-1}(1)| - K\Delta
\end{aligned}
$$

where the last equality is due to the rectangles being disjoint. The weak ε-approximation w.r.t. the uniform distribution $\mathcal{U}$ gives that $|\tilde{f} \neq f| \leq \varepsilon 2^n$, which we use to conclude. □

Combining Lemma 7.4 with Theorem 7.1, the proof of Theorem 7.3 boils down to showing that there are functions such that for every balanced rectangle r, the discrepancy $\mathrm{Disc}\,(f, r)$ can be suitably bounded, as shown in [BSW02].

7.3. Strong Approximations

In this section, we discuss some shortcomings of weak approximation and propose a stronger notion of approximation that avoids them. Let f_0 be the constant 0-function. We say that a function is *trivially weakly ε-approximable* (w.r.t. some distribution) if f_0 is a weak ε-approximation. Considering approximations w.r.t. the uniform distribution, it is easy to find classes of functions that are trivially weakly approximable.

Lemma 7.5. *Let $\varepsilon > 0$ and $0 \leq \alpha < 1$. Let $\mathcal{C}$ be a class of functions such that every function in $\mathcal{C}$ on n variables has at most $2^{\alpha n}$ models. Then there exists a constant n_0, such that any function from $\mathcal{C}$ on more than n_0 variables is trivially weakly ε-approximable w.r.t. the uniform distribution.*

Proof. Take $n_0 = \frac{1}{1-\alpha} \log(\frac{1}{\varepsilon})$ and choose f any function from $\mathcal{C}$ on $n > n_0$ variables. Then $|\{\mathbf{x} : f(\mathbf{x}) \neq f_0(\mathbf{x})\}| = |f^{-1}(1)| \leq 2^{\alpha n} < \varepsilon 2^n$. Therefore f_0 is a weak ε-approximation (w.r.t. the uniform distribution) of any function of $\mathcal{C}$ on sufficiently many variables. □

We remark that similar trivial approximation results can be shown for other distributions if the probability of a random assignment w.r.t. this distribution being a model is very small.

As a consequence, weak approximation makes no sense for functions with "few" (or "improbable") models. However such functions are often encountered, for example, random k-CNF with sufficiently many clauses are expected to have few models. Furthermore, even for functions with "many" models, one often studies encodings over larger sets of variables. For instance, when using Tseitin encoding to transform Boolean circuits into CNF, one introduces auxiliary variables that compute the value of sub-circuits under a given assignment. Generally, auxiliary variables are often used in practice since they reduce the representation size of functions, see e.g. [BHvMW09, Chapter 2]. The resulting encodings have more variables but most of the time the same number of models as the initial function. Consequently, they are likely to be trivially weakly approximable from Lemma 7.5. For these reasons we define a stronger notion of approximation.

Definition 7.6 (Strong approximation)**.** Let $\mathcal{D}$ be a distribution of the truth assignments to X and $\varepsilon > 0$. We say that $\tilde{f}$ is a *strong ε-approximation* of f (or strongly ε-approximates f) with respect to $\mathcal{D}$ if

$$\Pr_{\mathbf{x}\sim\mathcal{D}}\left[f(\mathbf{x}) \neq \tilde{f}(\mathbf{x})\right] \leq \varepsilon \Pr_{\mathbf{x}\sim\mathcal{D}}\left[f(\mathbf{x}) = 1\right].$$

When $\mathcal{D}$ is the uniform distribution $\mathcal{U}$, then the condition of strong approximability is equivalent to $|\{\mathbf{x} : f(\mathbf{x}) \neq \tilde{f}(\mathbf{x})\}| \leq \varepsilon |f^{-1}(1)|$. It is easy to see that strong approximation does not have the problem described in Lemma 7.5 for weak approximation. We also remark that strong approximation has been modeled to allow for efficient counting. In fact, a d-DNNF computing a strong ε-approximation of a function f allows approximate model counting for f with approximation factor ε.

Strong approximation has implicitly already been used in knowledge compilation. For instance it has been shown in [GKM+11] – although the authors use a different terminology – that for $\varepsilon > 0$, any Knapsack functions on n variables has a strong ε-approximation w.r.t. $\mathcal{U}$ that can be encoded by an OBDD of size polynomial in n and $1/\varepsilon$. The generalization to TAN [CT20] is also for strong approximations. These results are all the more significant since we know from [TNY97] that there exist threshold functions for which exact encodings by OBDD require size exponential in n.

Obviously, a strong approximation of f w.r.t. some distribution is also a weak approximation. Thus the statement of Theorem 7.3 can trivially be lifted to strong approximation. However the hard functions from Theorem 7.3 necessarily have sufficiently many models: if we are to consider only functions with few models, then they all are trivially weakly approximable. Yet we prove in the next section that there exist such functions whose exact encoding and strong ε-approximation encodings by d-DNNF require size exponential in n. Our proof follows the discrepancy method but relies on the following variant of Lemma 7.4 for strong approximation.

Lemma 7.7. *Let f be a Boolean function on n variables and let $\tilde{f}$ be a strong ε-approximation of f w.r.t. $\mathcal{U}$. Let $\tilde{f} = \bigvee_{k=1}^{K} r_k$ be a disjoint balanced rectangle cover of $\tilde{f}$ and assume that there is an integer $\Delta > 0$ such that $\mathrm{Disc}\,(f, r_k) \leq \Delta/2^n$ for all r_k. Then $K \geq (1-\varepsilon)|f^{-1}(1)|/\Delta$.*

Proof. The proof is essentially the same as for Lemma 7.4, differing only in the last lines where we use $|\tilde{f} \neq f| \leq \varepsilon |f^{-1}(1)|$ rather than $|\tilde{f} \neq f| \leq \varepsilon 2^n$. □

7.4. Large d-DNNFs for Strong Approximations

In this section, we show a lower bound for strong approximations of some functions that have weak approximations by small d-DNNFs. The functions we consider are characteristic functions of linear codes which we introduce now: a *linear code* of length n is a linear subspace of the vector space $\mathbb{F}_2^n$. Vectors from this subspace are called *code words*. A linear code is characterized by a *parity check matrix* H from $\mathbb{F}_2^{m \times n}$ as follows: a vector $\mathbf{x} \in \mathbb{F}_2^n$ is a code word if and only if $H\mathbf{x} = \mathbf{0}^m$ (operations are modulo 2 in $\mathbb{F}_2^n$). The *characteristic function* of a linear code is a Boolean function which accepts exactly the code words. Note that the characteristic function of a length n linear code of check matrix H has $2^{n-\mathrm{rk}(H)}$ models, where $\mathrm{rk}(H)$ denotes the rank of H. Following ideas developed in [DHJ+04], we focus on linear codes whose check matrices H have the following property: H is called *s-good* for some integer s if any submatrix obtained

by taking at least $n/3$ columns[1] from H has rank at least s. The existence of s-good matrices for $s = m - 1$ is guaranteed by the next lemma.

Lemma 7.8. [DHJ+04] *Let $m = n/100$ and sample a parity check matrix H uniformly at random from $\mathbb{F}_2^{m\times n}$. Then H is $(m-1)$-good with probability $1 - 2^{-\Omega(n)}$.*

Our interest in linear codes characterized by s-good matrices is motivated by another result from [DHJ+04] which states that the maximal size of any rectangle entailing the characteristic function of such a code decreases as s increases.

Lemma 7.9. [DHJ+04] *Let f be the characteristic function of a linear code of length n characterized by the s-good matrix H. Let r be a balanced rectangle such that $r \leq f$. Then $|r^{-1}(1)| \leq 2^{n-2s}$.*

Combining Lemmas 7.8 and 7.9 with Theorem 7.1, one gets the following result that was already observed in [29]:

Theorem 7.10. *There exists a class of linear codes $\mathcal{C}$ such that, for any code from $\mathcal{C}$ of length n, any d-DNNF encoding its characteristic function has size $2^{\Omega(n)}$.*

Proof. Let $m = n/100$. Lemma 7.8 ensures the existence of $(m-1)$-good matrices in $\mathbb{F}_2^{m\times n}$ for n large enough. Let $\mathcal{C}$ be the class of linear codes characterized by these matrices. Choose a code in $\mathcal{C}$ for the $(m-1)$-good matrix H and denote f its characteristic function, it has $2^{n-\mathrm{rk}(H)} \geq 2^{n-m}$ models. Let $\bigvee_{r\in R} r$ be a disjoint balanced rectangle cover of f. It holds that $|f^{-1}(1)| = \sum_{r\in R} |r^{-1}(1)|$ and we know from Lemma 7.9 that any r has $\leq 2^{n-2s} = 4 \times 2^{n-2m}$ models so $|f^{-1}(1)| \leq 4|R| \times 2^{n-2m}$. Using the lower bound on the number of models of f we obtain $|R| \geq \frac{1}{4}2^m$. Applying Theorem 7.1 finishes the proof. □

In the following, we will show that not only are characteristic functions hard to represent exactly as d-DNNF, they are even hard to strongly approximate.

Given the characteristic function f of a length n linear code of check matrix H, f has exactly $2^{n-\mathrm{rk}(H)}$ models. When $\mathrm{rk}(H)$ is at least a constant fraction of n, f satisfies the condition of Lemma 7.5, so for every $\varepsilon > 0$ and n large enough, f is trivially weakly ε-approximable (w.r.t. the uniform distribution). However we will show that any strong ε-approximation $\tilde{f}$ of f (w.r.t. the uniform distribution) only has d-DNNF encodings of size exponential in n.

To show this result, we will use the discrepancy method: we are going to find a bound on the discrepancy of f on any rectangle from a balanced disjoint rectangle cover of $\tilde{f}$. Then we will use the bound in Lemma 7.7 and combine the result with Theorem 7.1 to finish the proof.

Note that it is possible that a rectangle from a disjoint rectangle cover of $\tilde{f}$ makes no false positives on f. In fact, if this is the case for all rectangles in the cover, then $\tilde{f} \leq f$. In this case, lower bounds can be shown essentially as in the proof of Theorem 7.10. The

[1] Duris *et al.* [DHJ+04] limit to submatrices built from at least $n/2$ columns rather than $n/3$; however their result can easily be adapted.

more interesting case is thus that in which rectangles make false positives. In this case, we assume that no rectangle makes more false positives on f than it accepts models of f, because if such a rectangle r exists in a disjoint cover of $\tilde{f}$, then deleting r leads to a better approximation of f than $\tilde{f}$. Thus it is sufficient to consider approximations and rectangle covers in which all rectangles verify $|r^{-1}(1) \cap f^{-1}(1)| \geq |r^{-1}(1) \cap f^{-1}(0)|$.

Definition 7.11. Let r be a rectangle. A *core rectangle* (more succinctly a *core*) of r w.r.t. f is a rectangle r_{core} with the same partition as r such that

a) $r_{\text{core}} \leq f$ and $r_{\text{core}} \leq r$,

b) r_{core} is maximal in the sense that there is no r' satisfying a) such that $|r'^{-1}(1)| > |r_{core}^{-1}(1)|$.

Note that if $r \leq f$, then the only core rectangle of r is r itself. Otherwise r may have several core rectangles. We next state a crucial lemma on the relation of discrepancy and cores whose proof we defer to later parts of this section.

Lemma 7.12. *Let f be the characteristic function of some length n linear code, let r be a rectangle with more true positives than false positives on f, and let r_{core} be a core rectangle of r with respect to f, then*

$$\text{Disc}\,(f, r) \leq \frac{1}{2^n} |r_{core}^{-1}(1)|.$$

Lemma 7.12 says the following: consider a rectangle $r_{\text{core}} \leq f$ which is a core of a rectangle r. If r accepts more models of f than r_{core}, then for each additional such model r accepts at least one false positive. With Lemma 7.12, it is straightforward to show the main result of this section.

Theorem 7.13. *Let $0 \leq \varepsilon < 1$. There is a class of Boolean functions $\mathcal{C}$ such that any $f \in \mathcal{C}$ on n variables is trivially weakly ε-approximable w.r.t. $\mathcal{U}$ but any d-DNNF encoding a strong ε-approximation w.r.t. $\mathcal{U}$ has size $2^{\Omega(n)}$.*

Proof. Choose $\mathcal{C}$ to be the class of characteristic functions for length n linear codes characterized by $(m-1)$-good check matrices with $m = n/100$. Existence of these functions as n increases is guaranteed by Lemma 7.8. Let $\tilde{f}$ be a strong ε-approximation of $f \in \mathcal{C}$ w.r.t. $\mathcal{U}$ and let $\bigvee_{k=1}^{K} r_k$ be a rectangle cover of $\tilde{f}$. Combining Lemma 7.12 with Lemma 7.9, we obtain $\text{Disc}\,(f, r_k) \leq 2^{-n} 2^{n-2(m-1)}$. We then use Lemma 7.7 to get $K \geq (1-\varepsilon) 2^{2m-n} |f^{-1}(1)|/4$. The rank of the check matrix of f is at most m so $|f^{-1}(1)| \geq 2^{n-m}$ and $K \geq (1-\varepsilon) 2^m/4 = (1-\varepsilon) 2^{\Omega(n)}$. We use Theorem 7.1 to conclude. □

Note that Theorem 7.13 is optimal w.r.t. ε since for $\varepsilon = 1$ there is always the trivial approximation by the constant 0-function.

It remains to show Lemma 7.12 in the remainder of this section to complete the proof of Theorem 7.13. To this end, we make another definition.

Definition 7.14. Let (X_1, X_2) be a partition of the variables of f. A *core extraction operator* w.r.t. f is a mapping $\mathcal{C}_f$ that maps every pair (S_1, S_2) of sets of assignments over X_1 and X_2, respectively, to a pair (S'_1, S'_2) with

a) $S'_1 \subseteq S_1$ and $S'_2 \subseteq S_2$,

b) assignments from $S'_1 \times S'_2$ are models of f,

c) if f has no model in $S_1 \times S_2$, then $S'_1 = S'_2 = \emptyset$,

d) S'_1 and S'_2 are maximal in the sense that for every $S''_1 \subseteq S_1$ and every $S''_2 \subseteq S_2$ respecting the properties a), b) and c), we have $|S'_1||S'_2| \geq |S''_1||S''_2|$.

Intuitively S'_1 and S'_2 are the largest subsets one can extract from S_1 and S_2 such that assignments from $S'_1 \times S'_2$ are models of f. Note that, similarly to rectangle cores, the sets S'_1 and S'_2 are not necessarily uniquely defined. In this case, we assume that $\mathcal{C}_f$ returns an arbitrary pair with the required properties. One can show that core extraction operators yield core rectangles, as their name suggests.

Claim 7.15. *Let $r = \rho_1 \wedge \rho_2$ be a rectangle w.r.t. the partition (X_1, X_2) and denote $(A, B) = \mathcal{C}_f(\rho_1^{-1}(1), \rho_2^{-1}(1))$. Then the rectangle $\mathbf{1}_A \wedge \mathbf{1}_B$ is a core rectangle of r w.r.t. f.*

Proof. The rectangle $r_0 = \mathbf{1}_A \wedge \mathbf{1}_B$ is defined w.r.t. the same partition as r. We know justify that it is core rectangle for f, as defined in Definition 7.11:

a) $A \subseteq \rho_1^{-1}(1)$ and $B \subseteq \rho_2^{-1}(1)$ so $r_0 \leq r$ and all assignments from $A \times B$ are models of f so $r_0 \leq f$.

b) Assume r_0 is not maximal, that is, there exist $A' \subseteq \rho_1^{-1}(1)$ and $B' \subseteq \rho_2^{-1}(1)$ such that $r' = \mathbf{1}_{A'} \wedge \mathbf{1}_{B'} \leq f$ and $|r'^{-1}(1)| > |r_0^{-1}(1)|$. Then $|A'||B'| > |A||B|$, which contradicts the properties of $\mathcal{C}_f$. □

At this point, recall that f is the characteristic function of a linear code for a $m \times n$ check matrix H.

Claim 7.16. *Let $r = \rho_1 \wedge \rho_2$ be a rectangle w.r.t. the partition (X_1, X_2). Let $(A, B) = \mathcal{C}_f(\rho_1^{-1}(1), \rho_2^{-1}(1))$ and consider the core rectangle $r_{core} = \mathbf{1}_A \wedge \mathbf{1}_B$. Let $\overline{A} = \rho_1^{-1}(1) \setminus A$ and $\overline{B} = \rho_2^{-1}(1) \setminus B$. Then all assignments from $\overline{A} \times B$ and $A \times \overline{B}$ are false positives of r on f.*

Proof. Index the n columns of H with the variables in X (x_1 for column 1, x_2 for column 2, and so on). Let H_1 (resp. H_2) be the matrix obtained taking only the columns of H whose indices are in X_1 (resp. X_2). Obviously all vectors in $\overline{A} \times B$ and $A \times \overline{B}$ are models of r, but we will prove that they are not models of f. For every $\mathbf{a}' \in \overline{A}$ there is $\mathbf{b} \in B$ such that $H(\mathbf{a}', \mathbf{b}) = H_1\mathbf{a}' + H_2\mathbf{b} \neq \mathbf{0}^m$, otherwise the core rectangle would not be maximal. By definition of A and B, given $\mathbf{a} \in A$, for all $\mathbf{b} \in B$ we have $H(\mathbf{a}, \mathbf{b}) = H_1\mathbf{a} + H_2\mathbf{b} = \mathbf{0}^m$, so $H_2\mathbf{b}$ is constant over B. Therefore if $H_1\mathbf{a}' \neq H_2\mathbf{b}$ for *some* $\mathbf{b} \in B$ then $H_1\mathbf{a}' \neq H_2\mathbf{b}$ for *all* $\mathbf{b} \in B$. But then no vector from $\{\mathbf{a}'\} \times B$ can be a model of f and since $\mathbf{a}'$ has been chosen arbitrarily in $\overline{A}$, all vectors from $\overline{A} \times B$ are false positives. The case for $A \times \overline{B}$ follows analogously. □

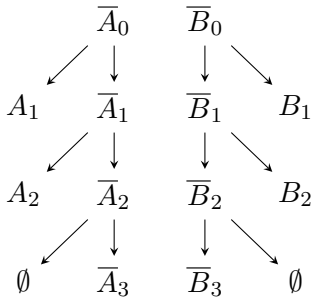


Figure 7.1.: An iterative core extraction with $l = 2$

For A and B defined as in Claim 7.16, we know that the assignments from $A \times B$ are models of f, and that those from $\overline{A} \times B$ and $A \times \overline{B}$ are not, but we have yet to discuss the case of $\overline{A} \times \overline{B}$. There may be additional models in this last set. The key to proving Lemma 7.12 is to iteratively extract core rectangles from $\mathbf{1}_{\overline{A}} \wedge \mathbf{1}_{\overline{B}}$ and control how many false positives are generated at each step of the iteration. To this end we define the collection $((A_i, B_i))_{i=0}^{l+1}$ as follows:

- $A_0 = \rho_1^{-1}(1)$ and $B_0 = \rho_2^{-1}(1)$,
- for $i \geq 1$, $(A_i, B_i) = \mathcal{C}_f(A_0 \setminus \bigcup_{j=1}^{i-1} A_j \ , \ B_0 \setminus \bigcup_{j=1}^{i-1} B_j)$,
- A_{l+1} and B_{l+1} are empty, but for any $i < l + 1$, neither A_i nor B_i is empty.

Denoting $\overline{A}_i := A_0 \setminus \bigcup_{j=1}^{i} A_j$ and $\overline{B}_i := B_0 \setminus \bigcup_{j=1}^{i} B_j$, we can write $(A_i, B_i) = \mathcal{C}_f(\overline{A}_{i-1}, \overline{B}_{i-1})$ (note that $\overline{A}_0 = A_0$ and $\overline{B}_0 = B_0$). Basically, we extract a core $(\mathbf{1}_{A_1} \wedge \mathbf{1}_{B_1})$ from r, then we extract a core $(\mathbf{1}_{A_2} \wedge \mathbf{1}_{B_2})$ from $(\mathbf{1}_{\overline{A}_1} \wedge \mathbf{1}_{\overline{B}_1})$, and so on until there is no model of f left in $\overline{A}_l \times \overline{B}_l$, in which case no core can be extracted from $(\mathbf{1}_{\overline{A}_l} \wedge \mathbf{1}_{\overline{B}_l})$ and $\mathcal{C}_f(\overline{A}_l, \overline{B}_l)$ returns $(\emptyset, \emptyset)$. The construction is illustrated in Figure 7.1.

Claim 7.17. *For any $i > 0$, all assignments from $F_i := (A_i \times \overline{B}_i) \cup (\overline{A}_i \times B_i)$ are false positives of r on f. Furthermore for every $i \neq j$ we have $F_i \cap F_j = \emptyset$.*

Proof. For the first part, it is clear from Claim 7.16 that assignments from $A_i \times \overline{B}_i$ and $\overline{A}_i \times B_i$ are false positives of $\mathbf{1}_{\overline{A}_{i-1}} \wedge \mathbf{1}_{\overline{B}_{i-1}}$ on f, and since $\mathbf{1}_{\overline{A}_{i-1}} \wedge \mathbf{1}_{\overline{B}_{i-1}} \leq r$, they are indeed false positives of r on f. For the second part, let $j > i > 0$, $F_i = (A_i \times \overline{B}_i) \cup (\overline{A}_i \times B_i)$ and $F_j = (A_j \times \overline{B}_j) \cup (\overline{A}_j \times B_j)$ are disjoint because both A_j and $\overline{A}_j$ are disjoint from A_i and both B_j and $\overline{B}_j$ are disjoint from B_i. □

Claim 7.18. *The function $\bigvee_{i=1}^{l}(\mathbf{1}_{A_i} \wedge \mathbf{1}_{B_i})$ is a disjoint rectangle cover of $r \wedge f$. Furthermore, if r is balanced, so are the rectangles from $\bigvee_{i=1}^{l}(\mathbf{1}_{A_i} \wedge \mathbf{1}_{B_i})$.*

Proof. By construction, the functions $(\mathbf{1}_{A_i} \wedge \mathbf{1}_{B_i})$ are rectangles with respect to the same partition as r. So if r is balanced, so are these rectangles.

For all i there is $(A_i \times B_i) \subseteq r^{-1}(1)$, so $\bigvee_{i=1}^{l}(\mathbf{1}_{A_i} \wedge \mathbf{1}_{B_i}) \leq r$. And by definition of $\mathcal{C}_f$, assignments from $A_i \times B_i$ are models of f, so $\bigvee_{i=1}^{l}(\mathbf{1}_{A_i} \wedge \mathbf{1}_{B_i}) \leq r \wedge f$.

To prove equality, assume that there exists $\mathbf{x}$ a model of r and f that is not a model of $\bigvee_{i=1}^{l}(\mathbf{1}_{A_i} \wedge \mathbf{1}_{B_i})$, that is, $\mathbf{x}$ does not belong to any $A_i \times B_i$ for $i > 0$. Then by Claim 3, $\mathbf{x}$ must be in $\overline{A}_l \times \overline{B}_l$ (figure 7.1 may help seeing this), but since $\overline{A}_l \times \overline{B}_l$ contains no models of f, this contradicts our assumption.

This proves that $\bigvee_{i=1}^{l}(\mathbf{1}_{A_i} \wedge \mathbf{1}_{B_i})$ is a rectangle cover of $r \wedge f$. The only thing left to prove is that the rectangles are disjoint. To see this, it is sufficient to observe that, for all $i > 1$, $A_i \subseteq \overline{A}_{i-1}$ which is disjoint from A_{i-1} and $B_i \subseteq \overline{B}_{i-1}$ which is disjoint from B_{i-1}. □

With Claim 7.17 and Claim 7.18, we can now prove Lemma 7.12.

Proof of Lemma 7.12. Claims 7.17 and 7.18 show that $\bigcup_{i=1}^{l}(A_i \times B_i) = r^{-1}(1) \cap f^{-1}(1)$ and $\bigcup_{i=1}^{l}\left((A_i \times \overline{B}_i) \cup (\overline{A}_i \times B_i)\right) \subseteq r^{-1}(1) \cap f^{-1}(0)$ and that these unions are disjoint. First we focus on the models of f covered by r.

$$|r^{-1}(1) \cap f^{-1}(1)| = \sum_{i=1}^{l} |A_i||B_i| = |r_{\text{core}}^{-1}(1)| + \sum_{i=2}^{l} |A_i||B_i|$$

where $r_{\text{core}} = \mathbf{1}_{A_1} \wedge \mathbf{1}_{B_1}$ is the first (therefore the largest) core rectangle extracted from r w.r.t. f. Now focus on the false positives of r on f

$$\begin{aligned} |r^{-1}(1) \cap f^{-1}(0)| &\geq \sum_{i=1}^{l} \left(|A_i||\overline{B}_i| + |\overline{A}_i||B_i|\right) \\ &\geq \sum_{i=1}^{l} \left(|A_i||B_{i+1}| + |A_{i+1}||B_i|\right) \end{aligned}$$

The maximality property of $\mathcal{C}_f$ implies $|A_i||B_i| \geq |A_{i+1}||B_{i+1}|$, and it follows that $|A_i||B_{i+1}| + |A_{i+1}||B_i| \geq |A_{i+1}||B_{i+1}|$. Thus

$$|r^{-1}(1) \cap f^{-1}(0)| \geq |r^{-1}(1) \cap f^{-1}(1)| - |r_{\text{core}}^{-1}(1)|.$$

By assumption, r accepts more models of f than false positives so $\text{Disc}(f, r) = (|r^{-1}(1) \cap f^{-1}(1)| - |r^{-1}(1) \cap f^{-1}(0)|)/2^n$ and the lemma follows directly. □

7.5. Conclusion

We have formalized and studied weak and strong approximation in knowledge compilation and shown functions that are hard to approximate by d-DNNFs with respect to these two notions. In particular, we have shown that strong approximations by d-DNNFs generally require exponentially bigger d-DNNF representations than weak approximations.

Let us sketch some directions for future research. One obvious question is to find for which classes of functions there *are* efficient algorithms computing approximations by d-DNNFs. In [CT20], it is shown that this is the case for certain Bayesian networks. It would be interesting to extend this to other settings to make approximation more applicable in knowledge compilation. Of particular interest are in our opinion settings

in which models are typically learned from data and thus inherently inexact, e.g. other forms of graphical models and neural networks.

Another question is defining and analyzing more approximation notions beyond weak and strong approximation. In fact, the latter was designed to allow approximate (weighted) counting as needed in probabilistic reasoning. Are there ways of defining notions of approximation that are useful for other problems, say optimization or entailment queries?

A more technical question is if one can show lower bounds for nondeterministic DNNF. In that setting, different rectangles may share the same false positives in which case our lower bound techniques break down. Are there approaches to avoid this problem?

8. Characterizing Tseitin-formulas with short regular resolution refutations

In this chapter, we give the details of [13], as discussed in Section 3.2. We remind the reader that the aim is to prove the following main result:

Theorem 8.1. *Let $T(G, c)$ be an unsatisfiable Tseitin-formula where G is a connected graph with maximum degree at most Δ. The length of the smallest regular resolution refutation of $T(G, c)$ is at least $2^{\Omega(\mathsf{tw}(G)/\Delta)}|V(G)|^{-1}$.*

We will start off with some preliminaries in Section 8.1. Then we will show how the proof complexity lower bound in Theorem 8.1 reduces to bounds on DNNF in Section 8.2. Before showing such a bound, we introduce a technique to show parameterized DNNF lower bounds in Section 8.3 which can be seen as an adaption of the results of Chapter 6 to allow easier results in a parameterized setting. In the rather technical Section 8.4 we will show how rectangles *split* the parity constraints in Tseitin-formulas in a certain sense and how this leads to the rectangles being small. Finally, Section 8.5 puts together all ingredients and gives the proof of Theorem 8.1.

8.1. Preliminaries

Notions on Graphs. We assume the that reader is familiar with the fundamentals of graph theory. For a graph G, we denote by $V(G)$ its vertices and by $E(G)$ its edges. For $v \in V(G)$, $E(v)$ denotes the edges incident to v and $N(v)$ its neighbors (v is not in $N(v)$). For a subset V' of $V(G)$ we denote by $G[V']$ the sub-graph of G induced by V'.

A binary tree whose leaves are in bijection with the edges of G is called a *branch decomposition*[1]. Each edge e of a branch decomposition T induces a partition of $E(G)$ into two parts as the edge sets that appear in the two connected components of T after deletion of e. The number of vertices of G that are incident to edges in both parts of this partition is the order of e, denoted by $order(e, T)$. The *branchwidth* of G, denoted by $bw(G)$, is defined as $bw(G) = \min_T \max_{e \in E(T)} order(e, T)$, where $\min_T$ is over all branch decompositions of G.

While it is convenient to work with branchwidth in our proofs, we state our main result with the more well-known *treewidth* $\mathsf{tw}(G)$ of a graph G. This is justified by the following well-known connection between the two measures.

Lemma 8.2. [HW17, Lemma 12] *If $bw(G) \geq 2$, then $bw(G) - 1 \leq \mathsf{tw}(G) \leq \frac{3}{2} bw(G)$.*

[1] We remark that often branch decompositions are defined as unrooted trees. However, it is easy to see that our definition is equivalent, so we use it here since it is more convenient in our setting.

A separator S in a connected graph G is defined to be a vertex set such that $G \setminus S$ is non-empty and not connected. A graph G is called 3-connected if and only if it has at least 4 vertices and, for every $S \subseteq V(G)$, $|S| \leq 2$, the graph $G \setminus S$ is connected.

Variables, assignments, v-trees. Boolean variables can have value 0 ($false$) or 1 ($true$). The notation ℓ_x refers to a literal for a variable x, that is, x or its negation $\overline{x}$. Given a set X of Boolean variables, $lit(X)$ denotes its set of literals. A truth assignment to X is a mapping $a : X \to \{0, 1\}$. If a_X and a_Y are assignments to *disjoint* sets of variables X and Y, then $a_X \cup a_Y$ denotes the combined assignment to $X \cup Y$. The set of assignments to X is denoted by $\{0, 1\}^X$. Let f be a Boolean function, we denote by $\mathsf{var}(f)$ its variables and by $\mathsf{sat}(f)$ its set of models, i.e., assignments to $\mathsf{var}(f)$ on which f evaluates to 1. A v-tree of X is a binary tree T whose leaves are labeled bijectively with the variables in X. A v-tree T of X induces a set of partitions (X_1, X_2) of X as follows: choose a vertex v of T, setting X_1 to contain exactly the variables in T that appear below v and $X_2 := X \setminus X_1$.

Tseitin-Formulas. Tseitin formulas are systems of parity constraints whose structure is determined by a graph. Let $G = (V, E)$ be a graph and let $c : V \to \{0, 1\}$ be a labeling of its vertices called a *charge function*. The Tseitin-formula $T(G, c)$ has for each edge $e \in E$ a Boolean variable x_e and for each vertex $v \in V$ a constraint $\chi_v : \sum_{e \in E(v)} x_e = c(v) \mod 2$. The Tseitin-formula $T(G, c)$ is then defined as $T(G, c) := \bigwedge_{v \in V} \chi_v$, i.e., the conjunction of the parity constraints for all $v \in V$. By $\overline{\chi_v}$ we denote the negation of χ_v, i.e., the parity constraint on $(x_e)_{e \in E(v)}$ with charge $1 - c(v)$.

Proposition 8.3. [Urq87, Lemma 4.1] *The Tseitin-formula $T(G, c)$ is satisfiable if and only if for every connected component U of G we have $\sum_{v \in U} c(v) = 0 \mod 2$.*

Proposition 8.4. [GI17, Lemma 2] *Let G be a graph with K connected components. If the Tseitin-formula $T(G, c)$ is satisfiable, then it has $2^{|E(G)|-|V(G)|+K}$ models.*

When conditioning the formula $T(G, c)$ on a literal $\ell_e \in \{x_e, \overline{x_e}\}$ for $e = ab$ in $E(G)$, the resulting function is another Tseitin formula $T(G, c)|\ell_e = T(G', c')$ where G' is the graph G without the edge e (so $G' = G - e$) and c' depends on ℓ_e. If $\ell_e = \overline{x_e}$ then c' equals c. If $\ell_e = x_e$ then $c' = c + 1_a + 1_b \mod 2$, where 1_v denotes the charge function that assigns 1 to v and 0 to all other variables.

Since we consider Tseitin-formulas in the setting of proof systems for CNF-formulas, we will assume in the following that they are encoded as CNF-formulas. In this encoding, every individual parity constraint χ_v is expressed as a CNF-formula F_v and $T(G, c) := \bigwedge_{v \in V} F_v$. Since it takes $2^{|E(v)|-1}$ clauses to write the parity constraint χ_v, each clause containing $E(v)$ literals, we make the standard assumption that $E(v)$ is bounded, i.e., there is a constant upper bound Δ on the degree of all vertices in G.

DNNF. A circuit over X in *negation normal form* (NNF) is a directed acyclic graph whose leaves are labeled with literals in $lit(X)$ or 0/1-constants, and whose internal nodes are labeled by $\vee$-gates or $\wedge$-gates. We use the usual semantics for the function computed

by (gates of) Boolean circuits. Every NNF can be turned into an equivalent NNF whose nodes have at most two successors in polynomial time. So we assume that NNF in this paper have only binary gates and thus define the size $|D|$ as the number of gates, which is then at most half the number of wires. Given a gate g, we denote by $\mathsf{var}(g)$ the variables for the literals appearing under g. When g is a literal input ℓ_x, we have $\mathsf{var}(g) = \{x\}$, and when it is a 0/1-input, we define $\mathsf{var}(g) = \emptyset$. A gate with two children g_l and g_r is called *decomposable* when $\mathsf{var}(g_l) \cap \mathsf{var}(g_r) = \emptyset$, and it is called *complete* (or *smooth*) when $\mathsf{var}(g_l) = \mathsf{var}(g_r)$. An NNF whose $\wedge$-gates are all decomposable is called a *decomposable NNF (DNNF)*. We call a DNNF *complete* when all its $\vee$-gates are complete. Every DNNF can be made complete in polynomial time. For every Boolean function f on finitely many variables, there exists a DNNF computing f.

When representing Tseitin-formulas by DNNF, we will use the following:

Lemma 8.5. *Let G be a graph and let c and c' be two charge functions such that $T(G, c)$ and $T(G, c')$ are satisfiable Tseitin-formulas. Then $T(G, c)$ can be computed by a DNNF of size s if and only if this is true for $T(G, c')$.*

Proof sketch. $T(G, c)$ can be transformed into $T(G, c')$ by substituting some variables by their negations, see [IRSS19, Proposition 26]. So every DNNF for $T(G, c)$ can be transformed into one for $T(G, c')$ by making the same substitutions. □

Branching programs. A branching program (BP) B is a directed acyclic graph with a single source, sinks that uniquely correspond to the values of a finite set Y, and whose inner nodes, called *decision nodes* are each labeled by a Boolean variable $x \in X$ and have exactly two output wires called the 0- and 1-wire pointing to two nodes respectively called its 0- and the 1-child. The variable x appears on a path in B if there is a decision node v labeled by x on that path. A truth assignment a to X induces a path in B which starts at the source and, when encountering a decision node for a variable x, follows the 0-wire (resp. the 1-wire) if $a(x) = 0$ (resp. $a(x) = 1$). The BP B is defined to compute the value $y \in Y$ on an assignment a if and only if the path of a leads to the sink labeled with y. We denote this value y as $B(a)$. Let $f : X \to Y$ be a function where X is a finite set of Boolean variables and Y any finite set. Then we say that B computes f if for every assignment $a \in \{0, 1\}^X$ we have $B(a) = f(a)$. We say that a node v in B computes a function g if the BP we get from B by deleting all nodes that are not reachable from v computes g.

Let $R \subseteq \{0, 1\}^X \times Y$ be a relation where Y is again finite. Then we say that a BP B computes R if for every assignment a we have that $(a, B(a)) \in R$. Let $T(G, c)$ be an unsatisfiable Tseitin-formula for a graph $G = (V, E)$. Then we define the two following relations: $\text{Search}_{T(G,c)}$ consists of the pairs (a, C) such that a is an assignment to $T(G, c)$ that does not satisfy the clause C of $T(G, c)$. The relation SearchVertex(G, c) consists of the pairs (a, v) such that a does not satisfy the parity constraint χ_v of a vertex $v \in V$. Note that $\text{Search}_{T(G,c)}$ and SearchVertex(G, c) both give a reason why an assignment a does not satisfy $T(G, c)$ but the latter is more coarse: SearchVertex(G, c) only gives a constraint that is violated while $\text{Search}_{T(G,c)}$ gives an exact clause that is not satisfied.

Regular Resolution. We only introduce some minimal notions of proof complexity here; for more details and references the reader is referred to the recent survey [BN21]. Let $C_1 = x \vee D_1$ and $C_2 = \overline{x} \vee D_2$ be two clauses such that D_1, D_2 contain neither x nor $\overline{x}$. Then the clause $D_1 \vee D_2$ is inferred by resolution of C_1 and C_2 on x. A resolution refutation of length s of a CNF-formula F is defined to be a sequence $C_1, \ldots, C_s$ such that C_s is the empty clause and for every $i \in [s]$ we have that C_i is a clause of F or it is inferred by resolution of two clauses C_j, C_ℓ such that $j, \ell < i$. It is well-known that F has a resolution refutation if and only if F is unsatisfiable.

To every resolution refutation $C_1, \ldots, C_s$ we assign a directed acyclic graph G as follows: the vertices of G are the clauses $\{C_i \mid i \in [s]\}$. Moreover, there is an edge C_jC_i in G if and only if C_i is inferred by resolution of C_j and some other clause C_ℓ on a variable x in the refutation. We also label the edge C_jC_i with the variable x. Note that there might be two pairs of clauses C_j, C_ℓ and $C_{j'}, C_{\ell'}$ such that resolution on both pairs leads to the same clause C_i. If this is the case, we simply choose one of them to make sure that all vertices in G have indegree at most 2. A resolution refutation is called *regular* if on every directed path in G every variable x appears at most once as a label of an edge. It is known that there is a resolution refutation of F if and only if a regular resolution refutation of F exists [DP60], but the latter are in general longer [AJPU07, Urq11].

In this paper, we will not directly deal with regular resolution proofs thanks to the following well-known result.

Theorem 8.6. [LNNW95] *For every unsatisfiable CNF-formula F, the length of the shortest regular resolution refutation of F is the size of the smallest* 1*-BP computing* Search_F*.*

Since in our setting, from an unsatisfied clause we can directly infer an unsatisfied parity constraint, we can use the following simple consequence.

Corollary 8.7. *For every unsatisfiable Tseitin-formula $T(G, c)$, the length of the shortest regular resolution refutation of $T(G, c)$ is at least the size of the smallest* 1*-BP computing* $\mathrm{SearchVertex}(G, c)$*.*

8.2. Reduction From Unsatisfiable to Satisfiable Formulas

To show our main result, we give a reduction from unsatisfiable to satisfiable Tseitin-formulas as in [IRSS19]. There it was shown that, given a 1-BP B computing the function $\mathrm{SearchVertex}(G, c)$ for an unsatisfiable Tseitin-formula $T(G, c)$, one can construct a 1-BP B' computing the function of a *satisfiable* Tseitin-formula $T(G, c^*)$ such that $|B'|$ is quasipolynomial in $|B|$. Then good lower bounds on the size of B' yield lower bounds for regular refutation by Corollary 8.7. To give tighter results, we give a version of the reduction from unsatisfiable to satisfiable Tseitin-formulas where the target representation for $T(G, c^*)$ is not 1-BP but the more succinct DNNF. This lets us decrease the size of the representation from pseudopolynomial to polynomial which, with tight lower bounds in the later parts of the paper, will yield Theorem 8.1.

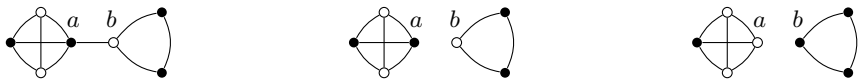


Figure 8.1.: The graphs of Example 8.9. On the left the graph G_k, in the middle the result after assigning 0 to x_e, on the right after assigning 1 to x_e.

Theorem 8.8. *Let $T(G, c)$ be an unsatisfiable Tseitin-formula where G is connected and let S be the length of its smallest resolution refutation. Then there exists for every satisfiable Tseitin-formula $T(G, c^*)$ a DNNF of size $O(S \times |V(G)|)$ computing it.*

In the proof of Theorem 8.8, we heavily rely on results from [IRSS19] in particular the notion of well-structuredness that we present in Section 8.2.1. In Section 8.2.2 we will then prove Theorem 8.8.

8.2.1. Well-structured branching programs for SearchVertex(G, c)

In a well-structured 1-BP computing SearchVertex(G, c), every decision node u_k for a variable x_e will compute SearchVertex(G_k, c_k) where G_k is a *connected* sub-graph of G containing the edge $e := ab$, and c_k is a charge function such that $T(G_k, c_k)$ is unsatisfiable. Since u_k deals with $T(G_k, c_k)$, its 0- and 1-successors u_{k_0} and u_{k_1} will work on $T(G_k, c_k)|\ell_e$ for $\ell_e = \overline{x_e}$ and $\ell_e = x_e$, respectively. $T(G_k, c_k)|\ell_e$ is a Tseitin-formula whose underlying graph is $G_k - e$ and whose charge function is c_k or $c_k + 1_a + 1_b \mod 2$ depending on ℓ_e. For convenience, we introduce the notation $\gamma_k(x_e) = c_k + 1_a + 1_b \mod 2$ and $\gamma_k(\overline{x_e}) = c_k$. Since G_k is connected, $G_k - e$ has at most two connected components. Let G_k^a and G_k^b denote the components of $G_k - e$ containing a and b, respectively. Note that $G_k^a = G_k^b$ when e is not a bridge of G_k. Let $\gamma_k^a(\ell_e)$ and $\gamma_k^b(\ell_e)$ denote the restriction of $\gamma_k(\ell_e)$ to the vertices of G_k^a and G_k^b, respectively. While the graph for $T(G_k, c_k)|\ell_e$ has at most two connected components, exactly one of them holds an odd total charge, so only the Tseitin-formula corresponding to that component is unsatisfiable. Well-structuredness states that u_{k_0} and u_{k_1} each deal with that unique connected component.

Example 8.9. *Consider the graph G_k shown on the left in Figure 8.1. Black nodes have charge 0 and white nodes have charge 1. The corresponding Tseitin-formula $T(G_k, c_k)$ is unsatisfiable because there is an odd number of white nodes. Let $e := ab$. Then $T(G_k, c_k)|\overline{x_e}$ is the Tseitin-formula for the graph $G_k - e$ with charges as shown in the middle of Figure 8.1. Note that $T(G_k, c_k)|\overline{x_e}$ is unsatisfiable because of the charges in the triangle component G_k^b. The repartition of charges for $T(G_k, c_k)|x_e$ illustrated on the right of Figure 8.1 shows that $T(G_k, c_k)|x_e$ is unsatisfiable because of the charges in the rombus component G_k^a. Well-structuredness will ensure that, if u_k computes* SearchVertex(G_k, c_k) *and decides x_e, then u_{k_0} computes* SearchVertex$(G_k^b, \gamma_k^b(\overline{x_e}))$ *and u_{k_1} computes* SearchVertex$(G_k^a, \gamma_k^a(x_e))$.

Definition 8.10. Let $T(G, c)$ be an unsatisfiable Tseitin-formula where G is a connected graph. A branching program B computing SearchVertex(G, c) is well-structured when, for all nodes u_k of B, there exists a connected subgraph G_k of G and a charge function c_k such that $T(G_k, c_k)$ is unsatisfiable, u_k computes SearchVertex(G_k, c_k) and

1. if u_k is the source, then $G_k = G$ and $c_k = c$,
2. if u_k is a sink corresponding to $v \in V(G)$, then $G_k = (\{v\}, \emptyset)$ and $c_k = 1_v$,
3. if u_k is a decision node for x_{ab} with 0- and 1- successors u_{k_0} and u_{k_1}, set $\ell_0 = \overline{x_{ab}}$ and $\ell_1 = x_{ab}$, then for all $i \in \{0,1\}$, $(G_{k_i}, c_{k_i}) = (G_k^a, \gamma_k^a(\ell_i))$ if $T(G_k^a, \gamma_k^a(\ell_i))$ is unsatisfiable, otherwise $(G_{k_i}, c_{k_i}) = (G_k^b, \gamma_k^b(\ell_i))$.

We remark that our definition is a slight simplification of that given by Itsykson et al. [IRSS19]. It can easily be seen that ours is implied by theirs (see Definition 11 and Proposition 16 in [IRSS19]).

Lemma 8.11. [IRSS19, Lemma 17] *Let $T(G,c)$ be an unsatisfiable Tseitin-formula where G is connected and let B be a 1-BP of minimal size*[2] *computing the relation* SearchVertex(G,c). *Then B is well-structured.*

8.2.2. Constructing DNNF from Well-structured branching programs

Similarly to Theorem 14 in [IRSS19], we give a reduction from a well-structured 1-BP for SearchVertex(G,c) to a DNNF computing a *satisfiable* formula $T(G,c^*)$.

Lemma 8.12. *Let G be a connected graph. Let $T(G,c^*)$ and $T(G,c)$ be Tseitin-formulas where $T(G,c^*)$ is satisfiable and $T(G,c)$ unsatisfiable. For every well-structured 1-BP B computing* SearchVertex(G,c) *there exists a DNNF of size $O(|B| \times |V(G)|)$ computing $T(G,c^*)$.*

Proof. Let $S = |B|$ and denote by $u_1, \ldots, u_S$ the nodes of B such that if u_j is a successor of u_i, then $j < i$ (thus u_S is the source of B). For every $i \in [S]$, the node u_i computes SearchVertex(G_i, c_i). We will show how to iteratively construct DNNF $D_1, \ldots, D_S$ such that, $D_1 \subseteq D_2 \subseteq \cdots \subseteq D_S$ and, for every $i \in [S]$,

$$\text{for all } v \in V(G_i), \text{ there is a gate } g_v \text{ in } D_i \text{ computing } T(G_i, c_i + 1_v). \qquad (*)$$

Observe that, since $T(G_i, c_i)$ is unsatisfiable, $T(G_i, c_i + 1_v)$ is satisfiable for any $v \in V(G_i)$. We show by induction on i how to construct D_i by extending D_{i-1} while respecting $(*)$.

For the base case, u_1 is a sink of B, so it computes SearchVertex$(G_v, 1_v)$ where $G_v := (\{v\}, \emptyset)$ for a vertex $v \in V(G)$. Thus we define D_1 as a single constant-1-node which indeed computes $T(G_v, 1_v + 1_v) = T(G_v, 0)$. So D_1 is a DNNF respecting $(*)$.

Now for the inductive case, suppose we have the DNNF D_{k-1} satisfying $(*)$. Consider the node u_k of B. If u_k is a sink of B, then we argue as for D_1 but since we already have the constant-1-node in D_{k-1} we define $D_k := D_{k-1}$.

Now assume that u_k is a decision node for the variable x_e with 0- and 1-successors u_{k_0} and u_{k_1}. Recall that u_k computes SearchVertex(G_k, c_k) and let $e = ab$. There are two cases. If e is not a bridge in G_k then $G_k^a = G_k^b = G_k - e$ and, by well-structuredness, u_{k_0} computes SearchVertex$(G_k - e, c_k)$ and u_{k_1} computes SearchVertex$(G_k - e, c_k + 1_a + 1_b)$. For every $v \in V(G_k)$, since $k_0, k_1 < k$, by induction there is a gate g_v^0 in D_{k_0} computing

[2] [IRSS19, Lemma 17] is for *locally minimal* 1-BP, which encompass minimal size 1-BP.

$T(G_k - e, c_k + 1_v)$ and a gate g_v^1 in D_{k_1} computing $T(G_k - e, c_k + 1_a + 1_b + 1_v)$. So for every $v \in V(G_k)$ we add to D_{k-1} an $\vee$-gate g_v whose left input is $\overline{x_e} \wedge g_v^0$ and whose right input is $x_e \wedge g_v^1$. By construction, g_v computes $T(G_k, c_k + 1_v)$ and the new $\wedge$-gates are decomposable since e is not an edge of $G_k - e$ and therefore x_e and $\overline{x_e}$ do not appear in D_{k_0} and D_{k_1}.

Now if $e = ab$ is a bridge in G_k, by well-structuredness, there exist $i \in \{0, 1\}$ and $\ell_e \in \{\overline{x_e}, x_e\}$ such that u_{k_i} computes SearchVertex$(G_k^a, \gamma_k^a(\ell_e))$ and $u_{k_{1-i}}$ computes SearchVertex$(G_k^b, \gamma_k^b(\overline{\ell_e}))$. We construct a gate g_v computing $T(G_k, c_k + 1_v)$ for each $v \in V(G_k)$. Assume, without loss of generality, that $v \in V(G_k^a)$, then

- $T(G_k, c_k + 1_v)|\overline{\ell_e} \equiv T(G_k^a, \gamma_k^a(\overline{\ell_e}) + 1_v) \wedge T(G_k^b, \gamma_k^b(\overline{\ell_e})) \equiv 0$
  (because of the second conjunct which is known to be unsatisfiable), and
- $T(G_k, c_k + 1_v)|\ell_e \equiv T(G_k^a, \gamma_k^a(\ell_e) + 1_v) \wedge T(G_k^b, \gamma_k^b(\ell_e))$

For the second item, since $k_0, k_1 < k$, by induction there is a gate g_v^i in D_{k_i} computing $T(G_k^a, \gamma_k^a(\ell_e) + 1_v)$ and there is a gate g_b^{1-i} in $D_{k_{1-i}}$ computing $T(G_k^b, \gamma_k^b(\overline{\ell_e}) + 1_b)$. But $\gamma_k(\ell_e) = \gamma_k(\overline{\ell_e}) + 1_a + 1_b \mod 2$, so $\gamma_k^b(\ell_e) = \gamma_k^b(\overline{\ell_e}) + 1_b \mod 2$, therefore g_b^{i-1} computes the formula $T(G_k^b, \gamma_k^b(\ell_e))$. So we add an $\wedge$-gate g_v whose left input is ℓ_e and whose right input is $s_v^i \wedge s_b^{1-i}$ and add it to D_{k-1}. Note that $\wedge$-gates are decomposable since G_k^a and G_k^b share no edge and therefore D_{k_0} and D_{k_1} are on disjoint sets of variables.

Let D_k be the circuit after all g_v have been added to D_{k-1}. It is a DNNF satisfying both $D_{k-1} \subseteq D_k$ and $(*)$.

It only remains to bound $|D_S|$. To this end, observe that when constructing D_k from D_{k-1} we add at most $3 \times |V_k|$ gates, so $|D_S|$ is at most $3(|V_1| + \cdots + |V_S|) = O(S \times |V(G)|)$. Finally, take any root of D_S and delete all gates not reached from it, the resulting circuit is a DNNF D computing a satisfiable Tseitin formula $T(G, c')$. We get a DNNF computing $T(G, c^*)$ using Lemma 8.5. □

Combining Corollary 8.7, Lemma 8.11 and Lemma 8.12 yields Theorem 8.8.

8.3. Adversarial Rectangle Bounds

In this section, we introduce the game we will use to show DNNF lower bounds for Tseitin formulas. It is based on combinatorial rectangles, a basic object of study from communication complexity.

Definition 8.13. A *(combinatorial) rectangle* for a variable partition (X_1, X_2) of a variables set X is defined to be a set of assignments of the form $R = A \times B$ where $A \subseteq \{0, 1\}^{X_1}$ and $B \subseteq \{0, 1\}^{X_2}$. The rectangle is called balanced when $\frac{|X|}{3} \leq |X_1|, |X_2| \leq \frac{2|X|}{3}$.

A rectangle on variables X may be seen as a function whose satisfying assignments are exactly the $a \cup b$ for $a \in A$ and $b \in B$, so we sometimes interpret rectangles as Boolean functions whenever it is convenient.

Definition 8.14. Let f be a Boolean function. A *balanced rectangle cover* of f is a collection $\mathcal{R} = \{R_1, \ldots, R_K\}$ of balanced rectangles on $\mathsf{var}(f)$, possibly for different partitions of $\mathsf{var}(f)$, such that f is equivalent to $\bigvee_{i=1}^{K} R_i$. The minimum number of rectangles in a balanced cover of f is denoted by $R(f)$.

Theorem 8.15. [27] *Let D be a DNNF computing a function f, then $R(f) \leq |D|$.*

When trying to show parameterized lower bounds with Theorem 8.15, one often runs into the problem that it is somewhat inflexible: the partitions of the rectangles in covers have to be balanced, but in parameterized applications this is often undesirable. Instead, to show good lower bounds, one wants to be able to partition in places that allow to cut in complicated subparts of the problem. This is e.g. the underlying technique in [Raz16]. To make this part of the lower bound proofs more explicit and the technique more reusable, we here introduce a refinement of Theorem 8.15.

We define the adversarial multi-partition rectangle cover game for a function f on variables X and a set $S \subseteq \mathsf{sat}(f)$ to be played as follows: two players, the cover player Charlotte and her adversary Adam, construct in several rounds a set $\mathcal{R}$ of combinatorial rectangles that cover the set S respecting f (that is, rectangles in $\mathcal{R}$ contain only models of f). The game starts with $\mathcal{R}$ as the empty set. Charlotte starts a round by choosing an input $a \in S$ and a v-tree T of X. Now Adam chooses a partition (X_1, X_2) of X induced by T. Charlotte ends the round by adding to $\mathcal{R}$ a combinatorial rectangle for this partition and respecting f that covers a. The game is over when S is covered by $\mathcal{R}$. The adversarial multi-partition rectangle complexity of f and S, denoted by $aR(f, S)$ is the minimum number of rounds in which Charlotte can finish the game, whatever the choices of Adam are. The following theorem gives the core technique for showing lower bounds later on. Due to space constraints, the proof is given in the full version.

Theorem 8.16. *Let D be a complete DNNF computing a function f and let $S \subseteq \mathsf{sat}(f)$. Then $aR(f, S) \leq |D|$.*

Proof. The proof of Theorem 8.16 uses the notion of proof trees of DNNF. Proof trees of a DNNF D are tree-like sub-circuits of D constructed iteratively as follows: we start from the root gate and add it to the proof tree. Whenever an $\wedge$-gate is met, both its child gates are added to the proof tree. Whenever a $\vee$-gate is met, exactly one child is is added to the proof tree. Each proof tree of D computes a conjunction of literals. By distributivity, the disjunction of the conjunctions computed by all proof trees of D computes the same function as D. When D is complete, every variable appears exactly once per proof tree, so every proof tree of a complete DNNF encodes a single model.

Let $X = \mathsf{var}(D)$. We iteratively delete vertices from D and construct rectangles. The approach is as follows: Charlotte chooses an assignment $a \in S$ not yet in any rectangle she constructed before and a proof tree T accepting a in D. By completeness of D, all variables of X appear exactly once in T. Charlotte constructs a v-tree of X from T by deleting negations on the leaves, contracting away nodes with a single child and forgetting the labels of all operation gates. Now Adam chooses a partition induced by T given by a subtree of T with root v. Note that v is a gate of C. Let $\mathsf{sat}(D, v) \subseteq \mathsf{sat}(f)$ be the assignments to X accepted by a proof tree of C passing through v, and observe that

$\mathsf{sat}(D, v)$ is a combinatorial rectangle $A \times B$ with $A \subseteq \{0,1\}^{\mathsf{var}(v)}$ and $B \subseteq \{0,1\}^{X \setminus \mathsf{var}(v)}$. Charlotte chooses the rectangle $\mathsf{sat}(D, v)$, deletes it from S and the game continues.

Note that the vertex v in the above construction is different for every iteration of the game: by construction, Charlotte never chooses an assignment a that is in any set $\mathsf{sat}(D, v)$ for a vertex v that has appeared before. Thus, no such v can appear in the proof tree of the chosen a. Consequently, a new vertex v is chosen for each assignment a that Charlotte chooses and thus the game will never last more than $|D|$ rounds. □

8.4. Splitting Parity Constraints

In this section, we will see that rectangles *split* parity constraints in a certain sense and show how this is reflected in in the underlying graph of Tseitin-formulas. This will be crucial in proving the DNNF lower bound in the next section with the adversarial multi-partition rectangle cover game.

8.4.1. Rectangles Induce Sub-Constraints for Tseitin-Formulas

Let R be a rectangle for the partition (E_1, E_2) of $E(G)$ such that $R \subseteq \mathsf{sat}(T(G, c))$. Assume that there is a vertex v of G incident to edges in E_1 and to edges in E_2, i.e., $E(v) = E_1(v) \cup E_2(v)$ where neither $E_1(v)$ not $E_2(v)$ is empty. We will show that R does not only respect χ_v, but it also respects a sub-constraint of χ_v.

Definition 8.17. Let χ_v be a parity constraint on $(x_e)_{e \in E(v)}$. A sub-constraint of χ_v is a parity constraint χ'_v on a non-empty proper subset of the variables of χ_v.

Lemma 8.18. *Let $T(G, c)$ be a satisfiable Tseitin-formula and let R be a rectangle for the partition (E_1, E_2) of $E(G)$ such that $R \subseteq \mathsf{sat}(T(G, c))$. If $v \in V(G)$ is incident to edges in E_1 and to edges in E_2, then there exists a sub-constraint χ'_v of χ_v such that $R \subseteq \mathsf{sat}(T(G, c) \wedge \chi'_v)$.*

Proof. Let $a_1 \cup a_2 \in R$ where a_1 is an assignment to E_1 and a_2 an assignment to E_2. Let $a_1(v)$ and $a_2(v)$ denote the restriction of a_1 and a_2 to $E_1(v)$ and $E_2(v)$, respectively. We claim that for all $a'_1 \cup a'_2 \in R$, we have that $a'_1(v)$ and $a_1(v)$ have the same parity, that is, $a_1(v)$ assigns an odd number of variables of $E_1(v)$ to 1 if and only if it is also the case for $a'_1(v)$. Indeed if $a_1(v)$ and $a'_1(v)$ have different parities, then so do $a_1(v) \cup a_2(v)$ and $a'_1(v) \cup a_2(v)$. So either $a_1 \cup a_2$ or $a'_1 \cup a_2$ falsifies χ_v, but both assignments are in R, so $a_1(v)$ and $a'_1(v)$ cannot have different parities as this contradicts $R \subseteq \mathsf{sat}(T(G, c))$. Let c_1 be the parity of $a_1(v)$, then we have that assignments in R must satisfy $\chi'_v : \sum_{e \in E_1(v)} x_e = c_1 \mod 2$, so $R \subseteq \mathsf{sat}(T(G, c) \wedge \chi'_v)$. □

Renaming χ'_v as χ^1_v and adopting notations from the proof, one sees that $\chi^1_v \wedge \chi_v \equiv \chi^1_v \wedge \chi^2_v$ where $\chi^2_v : \sum_{e \in E_2(v)} x_e = c(v) + c_1 \mod 2$. So R respects the formula $(T(G, c) - \chi_v) \wedge \chi^1_v \wedge \chi^2_v$ where $(T(G, c) - \chi_v)$ is the formula obtained by removing all clauses of χ_v from $T(G, c)$. In this sense, the rectangle is splitting the constraint χ_v into two subconstraints in disjoint variables. Since $\chi_v \equiv (\chi^1_v \wedge \chi^2_v) \vee (\overline{\chi}^1_v \wedge \overline{\chi}^2_v)$ it is plausible that potentially many models of χ_v are not in R. We show that this is true in the next section.

8.4.2. Vertex Splitting and Sub-constraints for Tseitin-Formulas

Let $v \in V(G)$ and let (N_1, N_2) be a proper partition of $N(v)$, that is, neither N_1 nor N_2 is empty. The graph G' we get by *splitting* v along (N_1, N_2) is defined as the graph we get by deleting v, adding two vertices v^1 and v^2, and connecting v^1 to all vertices in N_1 and v^2 to all vertices in N_2. We now show that splitting a vertex v in a graph G has the same effect as adding a sub-constraint of χ_v.

Lemma 8.19. *Let $T(G, c)$ be a Tseitin-formula. Let $v \in V(G)$ and let (N_1, N_2) be a proper partition of $N(v)$. Let c_1 and c_2 be such that $c_1 + c_2 = c(v) \mod 2$ and let $\chi_v^i : \sum_{u \in N_i} x_{uv} = c_i \mod 2$ for $i \in \{1, 2\}$ be sub-constraints of χ_v. Call G' the result of splitting v along (N_1, N_2) and set*

$$c'(u) := \begin{cases} c(u), & \text{if } u \in V(G) \setminus \{v\} \\ c_i, & \text{if } u = v^i, i \in \{1, 2\} \end{cases}$$

There is a bijection $\rho : \mathsf{var}(T(G, c)) \to \mathsf{var}(T(G', c'))$ acting as a renaming of the variables such that $T(G', c') \equiv (T(G, c) \wedge \chi_v^1) \circ \rho$.

Proof. Denote by $T(G, c) - \chi_v$ the formula equivalent to the conjunction of all χ_u for $u \in V(G) \setminus \{v\}$. Then $T(G, c) \wedge \chi_v^1 \equiv (T(G, c) - \chi_v) \wedge \chi_v^1 \wedge \chi_v^2$. The constraints χ_u for $u \in V(G) \setminus \{v\}$ appear in both $T(G', c')$ and in $T(G, c) - \chi_v$ and the sub-constraints χ_v^1 and χ_v^2 are exactly the constraints for v^1 and v^2 in $T(G', c')$ modulo the variable renaming ρ defined by $\rho(x_{uv}) = x_{uv^1}$ when $u \in N_1$, $\rho(x_{uv}) = x_{uv^2}$ when $u \in N_2$, and $\rho(x_e) = x_e$ when v is not incident to e. □

Intuitively, Lemma 8.19 says that splitting a vertex in G and adding sub-constraint are essentially the same operation. This allows us to compute the number of models of a Tseitin-formula to which a sub-constraint was added.

Lemma 8.20. *Let $T(G, c)$ be a satisfiable Tseitin-formula where G is connected. Define $T(G', c')$ as in Lemma 8.19. If G' is connected then $T(G', c')$ has $2^{|E(G)|-|V(G)|}$ models.*

Proof. By Proposition 8.3, $T(G', c')$ is satisfiable since $T(G, c)$ is satisfiable and

$$\sum_{u \in V(G')} c'(u) = \sum_{u \in V(G)} c(u) = 0 \mod 2.$$

Using Proposition 8.4 yields that $T(G', c')$ has $2^{|E(G')|-|V(G')|+1} = 2^{|E(G)|-|V(G)|}$ models. □

Lemma 8.21. *Let $T(G, c)$ be a satisfiable Tseitin-formula where G is connected. Let $\{v_1, \ldots, v_k\}$ be an independent set in G. For all $i \in [k]$ let (N_1^i, N_2^i) be a proper partition of $N(v_i)$ and let $\chi'_{v_i} : \sum_{u \in N_1^i} x_{uv_i} = c_i \mod 2$. If the graph obtained by splitting all v_i along (N_1^i, N_2^i) is connected, then the formula $T(G, c) \wedge \chi'_{v_1} \wedge \cdots \wedge \chi'_{v_k}$ has $2^{|E(G)|-|V(G)|-k+1}$ models.*

Proof. An easy induction based on Lemma 8.19 and Lemma 8.20. The induction works since, $\{v_1, \ldots, v_k\}$ being an independent set, the edges to modify by splitting v_i are still in the graph where $v_1, \ldots, v_{i-1}$ have been split. □

8.4.3. Vertex Splitting in 3-Connected Graphs

When we want to apply the results of the last sections to bound the size of rectangles, we require that the graph G remains connected after splitting vertices. This is obviously not true for all choices of vertex splits, but here we will see that if G is sufficiently connected, then we can always chose a large subset of any set of potential splits such that, after applying the split for this subset, G remains connected.

Lemma 8.22. *Let G be a 3-connected graph of and let $\{v_1, \dots, v_k\}$ be an independent set in G. For every $i \in [k]$ let (N_1^i, N_2^i) be a proper partition of $N(v_i)$. Then there is a subset S of $\{v_1, \dots, v_k\}$ of size at least $k/3$ such that the graph resulting from splitting all $v_i \in S$ along the corresponding (N_1^i, N_2^i) is connected.*

Proof. Let $C_1, \dots, C_r$ be the connected components of the graph G_1 that we get by splitting *all* v_i. If G_1 is connected, then we can set $S = \{v_1, \dots, v_k\}$ and we are done. So assume that $r > 1$ in the following. Now add for every $i \in [k]$ the edge (v_i^1, v_i^2). Call this edge set L (for *links*) and the resulting graph G_2. Note that G_2 is connected and for every edge set $E' \subseteq L$ we have that $G_2 \setminus E'$ is connected if and only if G is connected after splitting the vertices corresponding to the edges in E'. Denote by L_{in} the edges in L whose end points both lie in some component C_j and let $L_{out} := L \setminus L_{in}$.

We claim that for every C_j, at least three edges in L_{out} are incident to a vertex in C_j. Since G_2 is connected but the set C_j is a connected component of $G_2 \setminus L = G_1$, there must be at least one edge in L incident to a vertex in C_j. That vertex is by construction one of $v_1, \dots, v_k$, say it is v_i. Since $N_1^i \neq \emptyset$ and $N_2^i \neq \emptyset$, we have that v_i has a neighbor w in C_j and, $w \notin \{v_1, \dots, v_k\}$ since it is an independent set. Now let L_{out}^j be the edges in L_{out} that have an end point in C_j. Note that if we delete the vertices $S^j \subseteq \{v_1, \dots, v_k\}$ for which the edges in L_{out}^j were introduced in the construction of G_2, then a subset of C_j becomes disconnected from the rest of the graph (which is non-empty because there is at least one component different from C_j in G_2 which also contains a vertex not in $\{v_1, \dots, v_k\}$ by the same reasoning as before). But then, because G is 3-connected, there must be at least three edges in L_{out}^j. Let $k' := |L_{out}|$, then since $|L_{out}| = \frac{1}{2}\sum_{j=1}^{r} |L_{out}^j|$, we have that

$$r \leq \frac{2}{3}k'.$$

Now contract all components C_i in G_2 and call the resulting graph G_3. Note that G_3 is connected and that $E(G_3) = L_{out}$. Moreover, whenever $G_3 \setminus E^*$ is connected for some $E^* \subseteq L_{out}$, then G is connected after splitting the corresponding vertices. Choose any spanning tree T of G_3. Then $|E(T)| = r - 1$ and deleting $E^* := L_{out} \setminus E(T)$ leaves G_3 connected. Thus the graph G^* we get from G after splitting the vertices corresponding to E^* is connected. We have

$$|E^*| = |L_{out}| - |E(T)| = k' - (r-1) > \frac{k'}{3}.$$

Now observe that in G we can safely split all $k - k'$ vertices v_i that correspond to edges $v_i^1 v_i^2$ such that v_i^1 and v_i^2 lie in the same component of G_1 without disconnecting the

graph. Thus, overall we can split a set of size

$$k - k' + |E^*| > k - k' + \frac{k'}{3} \geq \frac{k}{3}$$

in G such that the resulting graph remains connected. □

8.5. DNNF Lower Bounds for Tseitin-Formulas

In this section, we use the results of the previous sections to show our lower bounds for DNNF computing Tseitin-formulas. To this end, we first show that we can restrict ourselves to the case of 3-connected graphs.

8.5.1. Reduction from Connected to 3-Connected Graphs

In [BK06], Bodlaender and Koster study how separators can be used in the context of treewidth. They call a separator S *safe for treewidth* if there exists a connected component of $G \setminus S$ whose vertex set V' is such that $\mathsf{tw}(G[S \cup V'] + \mathit{clique}(S)) = \mathsf{tw}(G)$, where $G[S \cup V'] + \mathit{clique}(S)$ is the graph induced on $S \cup V'$ with additional edges that pairwise connect all vertices in S.

Lemma 8.23. [BK06, Corollary 15] *Every separator of size 1 is safe for treewidth. When G has no separator of size 1, every separator of size 2 is safe for treewidth.*

Remember that a *topological minor* H of a G is a graph that can be constructed from G by iteratively applying the following operations:

- edge deletion,
- deletion of isolated vertices, or
- subdivision elimination: if $\deg(v) = 2$ delete v and connect its two neighbors.

Lemma 8.24. *Let H be a topological minor of G. If the satisfiable Tseitin-formula $T(G, 0)$ has a DNNF of size s, then so does $T(H, 0)$.*

Proof. Edge deletion corresponds to conditioning the variable by 0 so it cannot increase the size of a DNNF. Deletion of an isolated vertex does not change the Tseitin-formula. Finally, let e_1, e_2 be the edges incident to a vertex of degree 2. Since we assume that all charges $c(v)$ are 0, in every satisfying assignment, x_{e_1} and x_{e_2} take the same value. Thus we can simply forget the variable of x_{e_2} which does not increase the size of a DNNF [DM02]. □

Lemma 8.25. *Let G be a graph with treewidth at least 3. Then G has a 3-connected topological minor H with $\mathsf{tw}(H) = \mathsf{tw}(G)$.*

Proof. First we construct a topological minor of G with no separator of size 1 that preserves treewidth. Let $S = \{v\}$ be a separator of size 1 of G, then $G \setminus S$ has a connected component V' such that $G[S \cup V'] + \mathit{clique}(S) = G[S \cup V']$ has treewidth $\mathsf{tw}(G)$. Let $G' = G[S \cup V']$, then $\mathsf{tw}(G') = \mathsf{tw}(G)$. Observe that G' is a topological minor (remove all edges not in $G[S \cup V']$ thus isolating all vertices not in $S \cup V'$, which are then deleted) where S is no longer a separator. Repeat the construction until G' has no separator of size 1.

Now assume $S = \{u, v\}$ is a separator of G'. If V' are the vertices of a connected component of $G' \setminus S$, then there is a path from u to v in $G[S \cup V']$ since otherwise either $\{u\}$ or $\{v\}$ is a separator of size 1 of G'. Lemma 8.23 ensures that there is a connected component H' in $G' \setminus S$ such that $H := (V(H') \cup S, E(H') \cup \{uv\})$ has treewidth $\mathsf{tw}(H) = \mathsf{tw}(G') = \mathsf{tw}(G)$. Let us prove that H is topological minor of G'. Consider a connected component of $G' \setminus S$ distinct from H' with vertices V' and let P be a path connecting u to v in $G[S \cup V']$. Delete all edges of $G[S \cup V']$ not in P, then delete all isolated vertices in V' so that only P remains, finally use subdivision elimination to reduce P to a single edge uv. Repeat the procedure for all connected components of $G' \setminus S$ distinct from H', the resulting topological minor is $G[V(H') \cup S]$ with the (additional) edge uv, so H.

Repeat the construction until there are no separators of size 1 or size 2 left. Note that this process eventually terminates since the number of vertices decreases after every separator elimination. The resulting graph H is a topological minor of G of treewidth $\mathsf{tw}(G)$ without separators of size 1 or 2. Since $\mathsf{tw}(H) = \mathsf{tw}(G) \geq 3$, we have that H has at least 4 vertices, so H is 3-connected. □

8.5.2. Proof of the DNNF Lower Bound and of the Main Result

Lemma 8.26. *Let $T(G, c)$ be a satisfiable Tseitin-formula where G is a connected graph with maximum degree at most Δ. Any complete DNNF computing $T(G, c)$ has size at least $2^{\Omega(\mathsf{tw}(G)/\Delta)}$.*

Proof. By Lemma 8.5 we can set $c = 0$. By Lemmas 8.24 and 8.25 we can assume that G is 3-connected. We show that the adversarial multi-partition rectangle complexity is lower-bounded by 2^k for $k := \frac{2\mathsf{tw}(G)}{9\Delta}$. To this end, we will show that the rectangles that Charlotte can construct after Adam's answer are never bigger than $2^{|E(G)|-|V(G)|-k+1}$. Since $T(G, c)$ has $2^{|E(G)|-|V(G)|+1}$ models, the claim then follows.

So let Charlotte choose an assignment a and a v-tree T. Note that since the variables of $T(G, 0)$ are the edges of G, the v-tree T is also a branch decomposition of G. Now by the definition of branchwidth, Adam can choose a cut of T inducing a partition (E_1, E_2) of $E(G)$ for which there exists a set $V' \in V(G)$ of at least $bw(G) \geq \frac{2}{3}\mathsf{tw}(G)$ vertices incident to edges in E_1 and to edges in E_2.

G has maximum degree Δ so there is an independent set $V'' \subset V'$ of size at least $\frac{|V'|}{\Delta}$. Since G is 3-connected, by Lemma 8.22 there is a subset $V^* \subseteq V''$ of size at least $\frac{|V''|}{3} \geq \frac{2\mathsf{tw}(G)}{9\Delta} = k$ such that G remains connected after splitting of the nodes in V^* along the partition of their neighbors induced by the edges partition (E_1, E_2). Using

Lemma 8.18, we find that any rectangle R for the partition (E_1, E_2) respects a sub-constraint χ'_v for each $v \in V^*$. So R respects $T(G,0) \wedge \bigwedge_{v \in V^*} \chi'_v$. Finally, Lemma 8.21 shows that $|R| \leq 2^{|E(G)|-|V(G)|-k+1}$, as required. □

Theorem 8.1 is now a direct consequence of Theorem 8.8, Lemma 8.26 and Lemma 8.5

8.6. Conclusion

We have shown that the unsatisfiable Tseitin-formulas with polynomial length of regular resolution refutations are completely determined by the treewidth of their graphs. We did this by connecting lower bounds on these types of refutations to size bounds on DNNF representations of Tseitin-formulas. Moreover, we introduced a new two-player game that allowed us to show DNNF lower bounds.

Let us discuss some questions that we think are worth exploring in the future. First, it would be interesting to see if a $2^{\Omega(\mathsf{tw}(G))}$ lower bound for the refutation of Tseitin-formulas can also be shown for general resolution. In that case the length of resolution refutations would essentially be the same as that regular resolution refutations for Tseitin formulas. Note that this is somewhat plausible since other measures like space and width are known to be the same for the two proof systems for these formulas [GTT20].

Another question is the relation between knowledge compilation and proof complexity. As far as we are aware, our Theorem 8.8 is the first result that connects bounds on DNNF to such in proof complexity. It would be interesting to see if this connection can be strengthened to other classes of instances, other proof systems, representations from knowledge compilation and measures on proofs and representations, respectively.

9. Constant-Delay Enumeration for Document Spanners

In this chapter, we show the constant delay enumeration result for document spanners that was discussed in Section 3.5.3. Since constant delay algorithms have to satisfy extreme resource constraints, they are often quite technical and require many ingredients and tricks. Unfortunately, the same is true for the analysis of such algorithms. As a consequence, papers presenting results on constant delay are often quite long and technical. This is in particular true for the paper [20] which shows the result of this section: its journal version has 29 pages in a rather dense LaTeX-style; the related paper [25] has even 44 pages in the full arXiv version. I would have liked to present both these papers in this thesis, since both contain what I consider some very interesting technical ideas. However, this would have increased the length of this thesis to an unreasonable extent, so one of them had to go. After some thought, I decided to keep [20] with its result on document spanners in.

Even only keeping [20] would have taken quite some space if I had kept all of it, and so I decided to cut a part of it in this thesis, see Section 9.2 for details. The remaining parts give a complete proof of a slightly weakened result which allows to present many of the main ideas.

This chapter starts with some preliminaries before giving a formal statement of the main enumeration result in Section 9.2. We then introduce the main data structure used in the algorithm in Section 9.3 and show how to use it for enumeration in Section 9.4. In Section 9.5, we add the construction of an additional part of the preprocessing that the enumeration phase relies on. Finally, in Section 9.6 we present an implementation of our algorithm and validate it experimentally.

9.1. Preliminaries

Document spanners. We fix a finite alphabet Σ. A *document* $d = d_0 \cdots d_{n-1}$ is just a word over Σ. A *span* of d is a pair $[i, j\rangle$ with $0 \leq i \leq j \leq |d|$ which represents a substring (contiguous subsequence) of d starting at position i and ending at position $j - 1$. To describe the possible results of an information extraction task, we will use a finite set $\mathcal{V}$ of variables, and define a result as a *mapping* from these variables to spans of the input document. Following [FRU+18, MRV18] but in contrast to [FKRV15], we will not require mappings to assign all variables: formally, a *mapping* of $\mathcal{V}$ on d is a function μ from some domain $\mathcal{V}' \subseteq \mathcal{V}$ to spans of d. We define a *document spanner* to be a function assigning to every input document d a set of mappings, which denotes the set of results of the extraction task on the document d.

Variable-set automata. We will represent document spanners using *variable-set automata* (or *VAs*). The transitions of a VA can carry letters of Σ or *variable markers*, which are either of the form $x\vdash$ for a variable $x \in \mathcal{V}$ (denoting the start of the span assigned to x) or $\dashv x$ (denoting its end). Formally, a *variable-set automaton* $\mathcal{A}$ (or VA) is then defined to be an automaton $\mathcal{A} = (Q, q_0, F, \delta)$ where the transition relation δ consists of *letter transitions* of the form (q, a, q') for $q, q' \in Q$ and $a \in \Sigma$, and of *variable transitions* of the form $(q, x\vdash, q')$ or $(q, \dashv x, q')$ for $q, q' \in Q$ and $x \in \mathcal{V}$. A *configuration* of a VA is a pair (q, i) where $q \in Q$ and i is a position of the input document d. A *run* σ of $\mathcal{A}$ on d is then a sequence of configurations

$$(q_0, i_0) \xrightarrow{\sigma_1} (q_1, i_1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_m} (q_m, i_m)$$

where $i_0 = 0$, $i_m = |d|$, and where for every $1 \leq j \leq m$, one of the following holds:

- The label σ_j is a letter of Σ, we have $i_j = i_{j-1} + 1$, we have $d_{i_{j-1}} = \sigma_j$, and (q_{j-1}, σ_j, q_j) is a letter transition of $\mathcal{C}$;
- The label σ_j is a variable marker, we have $i_j = i_{j-1}$, and (q_{j-1}, σ_j, q_j) is a variable transition of $\mathcal{C}$. In this case we say that the variable marker σ_j is *read* at position i_j.

As usual, we say that a run is *accepting* if $q_m \in F$. A run is *valid* if it is accepting, every variable marker is read at most once, if an open marker $x\vdash$ is read at a position i then the corresponding close marker $\dashv x$ is read at a position i' with $i \leq i'$, and if $x\vdash$ is not read then $\dashv x$ is not read either. Each valid run defines a mapping on the domain $\mathcal{V}'$ of the variables for which the run has read some markers: specifically, each variable $x \in \mathcal{V}'$ is mapped to the span $[i, i'\rangle$ such that $x\vdash$ is read at position i and $\dashv x$ is read at position i'. The *document spanner* of the VA $\mathcal{C}$ is then the function that assigns to every document d the set of mappings defined by the valid runs of $\mathcal{C}$ on d: note that the same mapping can be defined by multiple different runs, and note that the different runs may have different domains. The task studied in this paper is the following: given a VA $\mathcal{C}$ and a document d, enumerate *without duplicates* the mappings that are assigned to d by the document spanner of $\mathcal{C}$. The enumeration must write each mapping as a set of pairs (m, i) where m is a variable marker and i is a position of d, each set being written as a sequence in some arbitrary order. We will say that a set of pairs of markers and positions is *valid* when every marker occurs at most once in the set, if an open marker $x\vdash$ occurs in the set as $(x\vdash, i)$ then the set also contains $(\dashv x, i')$ with $i < i'$, and if $x\vdash$ does not occur in the set then neither does $\dashv x$. Thus, the results of the enumeration are always valid in this sense. Note that we will often abuse notation and identify the function representation of mappings defined above with this representation as a set of pairs which is valid.

Sequential VAs. We cannot hope to efficiently enumerate the mappings of arbitrary VAs because it is already NP-complete to decide if, given a VA $\mathcal{A}$ and a document d, there are any valid runs of $\mathcal{A}$ on d [Fre17]. For this reason, we will restrict ourselves to so-called *sequential* VAs [MRV18]. A VA $\mathcal{C}$ is *sequential* if for every document d, every accepting run of $\mathcal{C}$ of d is also valid: this implies that the document spanner of $\mathcal{C}$

can simply be defined following the accepting runs of $\mathcal{C}$. If we are given a VA, then we can test in NL whether it is sequential [MRV18, Proposition 5.5], and otherwise we can convert it to an equivalent sequential VA (i.e., that defines the same document spanner) with an unavoidable exponential blowup in the number of variables (not in the number of states), using existing results:

Proposition 9.1. *Given a VA $\mathcal{C}$ on variable set $\mathcal{V}$, letting $k := |\mathcal{V}|$ and r be the number of states of $\mathcal{C}$, we can compute an equivalent sequential VA $\mathcal{C}'$ with $3^k r$ states. Conversely, for any $k \in \mathbb{N}$, there exists a VA $\mathcal{C}_k$ with 1 state on a variable set with k variables such that any sequential VA equivalent to $\mathcal{C}_k$ has at least 3^k states.*

Proof. This can be shown exactly like [Fre17, Proposition 12] and [Fre19, Proposition 3.9]. In short, the upper bound is shown by modifying $\mathcal{C}$ to remember in the automaton state which variables have been opened or closed, and by re-wiring the transitions to ensure that the run is valid: this creates 3^k copies of every state because each variable can be either unseen, opened, or closed. For the lower bound, [Fre19, Proposition 3.9] gives a VA for which any equivalent sequential VA must remember the status of all variables in this way. □

All VAs studied in this work will be sequential, and we will further assume that they are *trimmed* in the sense that for every state q there is a document d and an accepting run of the VA where the state q appears. This condition can be enforced in linear time on any sequential VA: we do a graph traversal to identify the accessible states (the ones that are reachable from the initial state), we do another graph traversal to identify the co-accessible states (the ones from which we can reach a final state), and we remove all states that are not accessible or not co-accessible. We will implicitly assume that all sequential VAs have been trimmed, which implies that they cannot contain any cycle of variable transitions (as such a cycle would otherwise appear in a run, which would not be valid).

Extended VAs. We will first prove our results for a variant of sequential VAs introduced by [FRU$^+$18], called sequential *extended VAs*. An extended VA on alphabet Σ and variable set $\mathcal{V}$ is an automaton $\mathcal{A} = (Q, q_0, F, \delta)$ where the transition relation δ consists of *letter transitions* as before, and of *extended variable transitions* (or *ev-transitions*) of the form (q, M, q') where M is a possibly empty set of variable markers. Intuitively, on ev-transitions, the automaton reads multiple markers at once. Formally, a *run* σ of $\mathcal{A}$ on $d = d_0 \cdots d_{n-1}$ is a sequence of configurations (defined like before) where letter transitions and ev-transitions alternate:

$$(q_0, 0) \xrightarrow{M_0} (q'_0, 0) \xrightarrow{d_0} (q_1, 1) \xrightarrow{M_1} (q'_1, 1) \xrightarrow{d_1} \cdots \xrightarrow{d_{n-1}} (q_n, n) \xrightarrow{M_n} (q'_n, n)$$

where (q'_i, d_i, q_{i+1}) is a letter transition of $\mathcal{C}$ for all $0 \leq i < n$, and (q_i, M_i, q'_i) is an ev-transition of $\mathcal{C}$ for all $0 \leq i \leq n$ where M_i is the set of variable markers *read* at position i. Accepting and valid runs are defined like before, and the extended VA is sequential if all accepting runs are valid, in which case its document spanner is defined like before.

Our definition of extended VAs is slightly different from [FRU+18] because we allow ev-transitions that read the empty set to change the automaton state. This allows us to make a small additional assumption to simplify our proofs: we require that the states of extended VAs are partitioned between *ev-states*, from which only ev-transitions originate (i.e., the q_i above), and *letter-states*, from which only letter transitions originate (i.e., the q'_i above); and we impose that the initial state is an ev-state and the final states are all letter-states. Note that transitions reading the empty set move from an ev-state to a letter-state, like all other ev-transitions. Our requirement can be imposed in linear time on any extended VA, by rewriting each state to one letter-state and one ev-state, and re-wiring the transitions and changing the initial/final status of states appropriately. This rewriting preserves sequentiality and guarantees that any path in the rewritten extended VA must alternate between letter transitions and ev-transitions. Hence, we implicitly make this assumption on all extended VAs from now on.

Example 9.2. *The top of Figure 9.1 represents a sequential extended VA $\mathcal{C}_0$ to extract email addresses. To keep the example readable, we simply define them as words (delimited by a space or by the beginning or end of document) which contain one at-sign "@" preceded and followed by a non-empty sequence of non-"@" characters. In the drawing of $\mathcal{C}_0$, the initial state q_0 is at the left, and the states q_{10} and q_{12} are final. The transitions labeled by Σ represent a set of transitions for each letter of Σ, and the same holds for Σ' which we define as $\Sigma' := \Sigma \setminus \{@, \_\}$.*

It is easy to see that, on any input document d, there is one mapping of $\mathcal{C}_0$ on d per email address contained in d, which assigns the markers $x\vdash$ and $\dashv x$ to the beginning and end of the email address, respectively. In particular, $\mathcal{C}_0$ is sequential, because any accepting run is valid. Note that $\mathcal{C}_0$ happens to have the property that each mapping is produced by exactly one accepting run, but our results in this paper do not rely on this property.

Matrix multiplication. The complexity bottleneck for some of our results will be the complexity of multiplying two Boolean matrices, which is a long-standing open problem, see e.g. [Gal12] for a recent discussion. When stating our results, we will often denote by $2 \leq \omega \leq 3$ an exponent for Boolean matrix multiplication: this is a constant such that the product of two r-by-r Boolean matrices can be computed in time $O(r^\omega)$. For instance, we can take $\omega := 3$ if we use the naive algorithm for Boolean matrix multiplication, and it is obvious that we must have $\omega \geq 2$. The best known upper bound is currently $\omega < 2.3728639$, see [Gal14].

9.2. The Main Enumeration Result

Now that we have all preliminaries and definitions in place, we can formulate the main result of this chapter.

Theorem 9.3. *Let $2 \leq \omega \leq 3$ be an exponent for Boolean matrix multiplication. Let $\mathcal{C}$ be a sequential VA with variable set $\mathcal{V}$ and with state set Q, and let d be an input document.*

We can enumerate the mappings of $\mathcal{C}$ on d with preprocessing time in $O((|Q|^{\omega+1}+|\mathcal{C}|)\times|d|)$ and with delay $O(|\mathcal{V}| \times (|Q|^2 + |\mathcal{C}| \times |\mathcal{V}|^2))$, i.e., linear preprocessing and constant delay in the input document, and polynomial preprocessing and delay in the input VA.

To keep the length of this chapter manageable, we will in fact only show a simplified version of Theorem 9.3 here. Instead of general sequential VAs, we will only show it for extended sequential VAs. The extension to the more general model is done in [20] following the same ideas but has one additional ingredient: we will have to navigate certain graphs, the mapping DAGs introduced in the next section, whose paths correspond to the outputs we want to enumerate. Unfortunately, there will be several paths that differ locally in a certain sense which give the same output and which we are not allowed to enumerate several times. The crucial idea is then to use a backtracking algorithm, also called flashlight search in the enumeration literature, as a subroutine to locally enumerate partial solutions and merge them with the algorithm for the case of extended VAs to get an algorithm for general sequential VAs. This requires some technical care to do correctly, and we refer the interested reader to [20, 1] for the details.

Our result Theorem 9.3 implies analogous results for all spanner formalisms that can be translated to sequential VAs. In particular, spanners are not usually written as automata by users, but instead given in a form of regular expressions called *regex-formulas*, see [FKRV15] for exact definitions. As we can translate sequential regex-formulas to sequential VAs in linear time [FKRV15, FKP18, MRV18], our results imply that we can also evaluate them.

Another direct application of our result is for so-called *regular spanners*, which are unions of conjunctive queries (UCQs) posed on regex-formulas, i.e., the closure of regex-formulas under union, projection and joins. We again point the reader to [FKRV15, FKP18] for the full definitions. As such UCQs can in fact be evaluated by VAs, our result also implies tractability for such representations, as long as we only perform a bounded number of joins.

9.3. Computing Mapping DAGs for Extended VAs

As discussed before, in this thesis we will restrict ourselves to studying extended VAs, which are easier to work with because the set of markers that can be assigned at every position is explicitly written as the label of a single transition. We accordingly show Theorem 9.3 for the case of extended VAs in Sections 9.3–9.5. The case of non-extended VAs is discussed in [20].

Mapping DAGs. To show Theorem 9.3 for extended VAs, we will reduce the problem of enumerating the mappings captured by $\mathcal{C}$ to that of enumerating path labels in a special kind of directed acyclic graph (DAG), called a *mapping DAG*. This DAG is intuitively a variant of the product of $\mathcal{C}$ and of the document d, where we represent simultaneously the position in the document and the corresponding state of $\mathcal{C}$. We will no longer care in the mapping DAG about the labels of letter transitions, so we will erase these labels and call these transitions *ϵ-transitions*. As for the ev-transitions, we will extend their labels

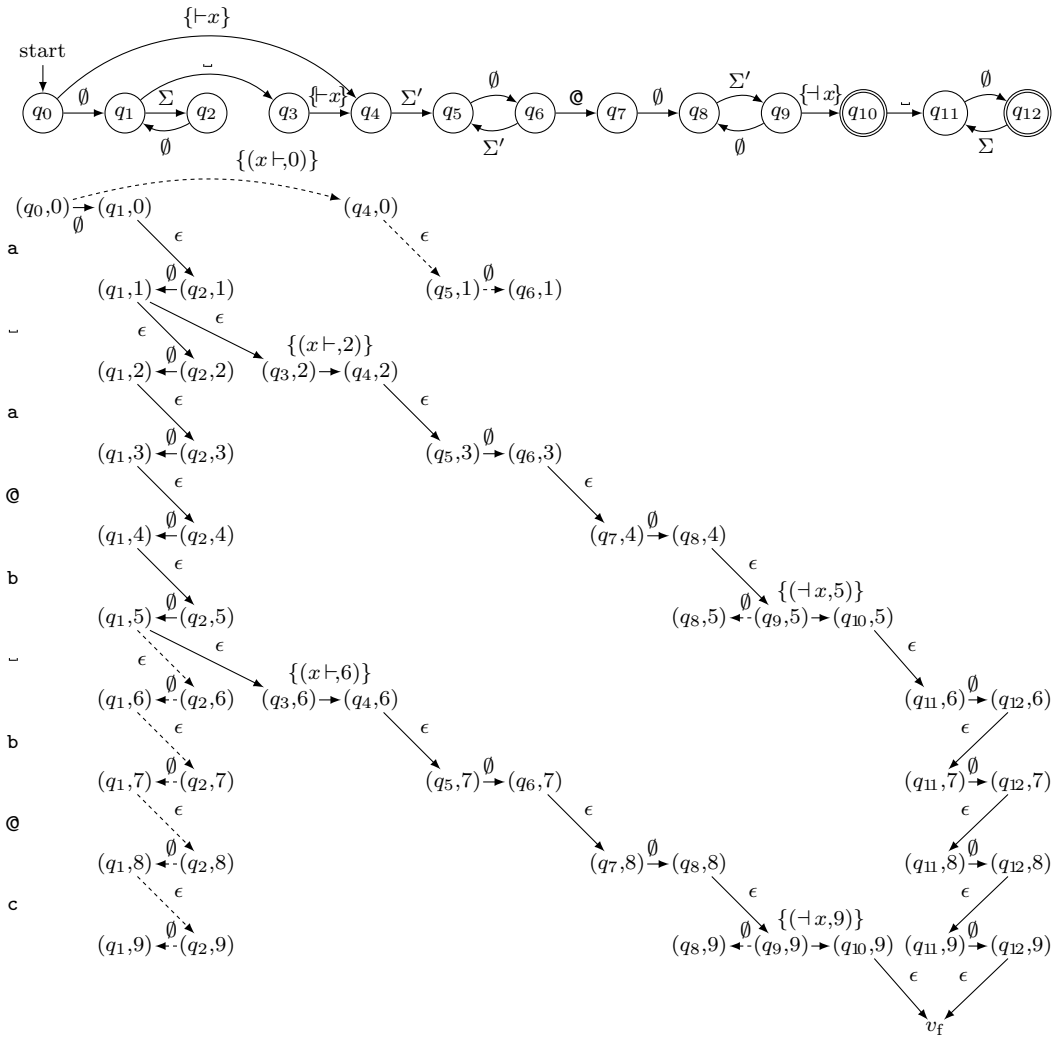


Figure 9.1.: Example sequential extended VA $\mathcal{C}_0$ to extract e-mail addresses (see Example 9.2) and example mapping DAG on an example document (see Examples 9.6, 9.9, 9.10, and 9.13).

to indicate the position in the document in addition to the variable markers. We first give the general definition of a mapping DAG:

Definition 9.4. A *mapping DAG* consists of a set V of *vertices*, an *initial vertex* $v_0 \in V$, a *final vertex* $v_\mathrm{f} \in V$, and a set of *edges* E where each edge (s, x, t) has a *source vertex* $s \in V$, a *target vertex* $t \in V$, and a *label* x that may be ϵ (in which case we call the edge an *ϵ-edge*) or a finite (possibly empty) set of pairs (m, i), where m is a variable marker and i is a position. These edges are called *marker edges*. We require that the graph (V, E) is acyclic. We say that a mapping DAG is *normalized* if every path in the mapping DAG alternates between marker edges and ϵ-edges, every path starting at the initial vertex starts with a marker edge, and every path ending at the final vertex ends with an ϵ-edge.

The *pre-mapping* $\mu(\pi)$ of a path π in the mapping DAG is the union of labels of the marker edges of π: we require of any mapping DAG that, for every path π, this union is disjoint, and that for every path π from v_0 to v_f, the pre-mapping $\mu(\pi)$ is valid, i.e., it corresponds to a mapping. Given a set U of vertices of G, we write $\mathcal{M}(U)$ for the set of pre-mappings of paths from a vertex of U to the final vertex; note that the same pre-mapping may be captured by multiple different paths. The set of pre-mappings *captured* by G is then $\mathcal{M}(G) := \mathcal{M}(\{v_0\})$; all of these are mappings, i.e., they are valid.

Intuitively, the ϵ-edges will correspond to letter transitions of $\mathcal{C}$ (with the letter being erased, i.e., replaced by ϵ), and marker edges will correspond to ev-transitions: their labels are a possibly empty finite set of pairs of a variable marker and position, describing which variables have been assigned during the transition. We now explain how we construct a DAG from $\mathcal{C}$ and from a document d, which we call the *product DAG* of $\mathcal{C}$ and d, and which we will show to be a mapping DAG:

Definition 9.5. Let $\mathcal{C} = (Q, q_0, F, \delta)$ be a sequential extended VA and let $d = d_0 \cdots d_{n-1}$ be an input document. The *product DAG* of $\mathcal{C}$ and d is the DAG whose vertex set is $Q \times \{0, \ldots, n\} \cup \{v_\mathrm{f}\}$ with $v_\mathrm{f} := (\bullet, n+1)$ for some fresh value $\bullet$. Its edges are:

- For every letter-transition (q, a, q') in δ, for every $0 \leq i < |d|$ such that $d_i = a$, there is an ϵ-edge from (q, i) to $(q', i+1)$;
- For every ev-transition (q, M, q') in δ, for every $0 \leq i \leq |d|$, there is a marker edge from (q, i) to (q', i) labeled with the (possibly empty) set $\{(m, i) \mid m \in M\}$.
- For every final state $q \in F$, an ϵ-edge from (q, n) to v_f.

The initial vertex of the product DAG is $(q_0, 0)$ and the final vertex is v_f.

Note that, contrary to [FRU+18], we do not contract the ϵ-edges but keep them throughout our algorithm.

Example 9.6. *The product DAG of our example sequential extended VA $\mathcal{C}_0$ and of the example document* `a␣a@b␣b@c` *is shown on Figure 9.1, with the document being written at the left from top to bottom. The initial vertex of the DAG is $(q_0, 0)$ at the top left*

and its final vertex is v_f at the bottom. We draw marker edges horizontally, and ϵ-edges diagonally. To simplify the example, we only draw the parts of the DAG that are reachable from the initial vertex. Edges are dashed when they cannot be used to reach the final vertex.

It is easy to see that this construction satisfies the definition:

Claim 9.7. *The product DAG of $\mathcal{C}$ and d is a normalized mapping DAG.*

Proof. It is immediate that the product DAG is indeed acyclic, because the second component is always nondecreasing, and an edge where the second component does not increase (corresponding to an ev-transition of the VA) must be followed by an edge where it does (corresponding to a letter-transition of the VA). What is more, we claim that no path in the product DAG can include two edges whose labels contain the same pair (m, i), so that the unions used to define the mappings of the mapping DAG are indeed disjoint. To see this, consider a path from an edge $((q_1, i_1), M_1, (q_1', i_1))$ to an edge $((q_2, i_2), M_2, (q_2', i_2))$ where $M_1 \neq \epsilon$ and $M_2 \neq \epsilon$, we have $i_1 < i_2$ and M_1 and M_2 are disjoint because all elements of M_1 have i_1 as their first component, and all elements of M_2 have i_2 as their first component. Further, the product DAG is also normalized because $\mathcal{C}$ is an extended VA that we have preprocessed to distinguish letter-states and ev-states. □

Further, the product DAG clearly captures what we want to enumerate. Formally:

Claim 9.8. *The set of mappings of $\mathcal{C}$ on d is exactly the set of mappings $\mathcal{M}(G)$ captured by the product DAG G.*

Proof. This is immediate as there is a clear bijection between accepting runs of $\mathcal{C}$ on d and paths from the initial vertex of G to its final vertex, and this bijection ensures that the label of the path in G is the mapping corresponding to that accepting run. □

Example 9.9. *The set of mappings captured by the example product DAG on Figure 9.1 is*

$$\{\ \{(x\vdash, 2), (\dashv x, 5)\}, \{(x\vdash, 6), (\dashv x, 9)\}\ \}$$

and this is indeed the set of mappings of the example extended VA $\mathcal{C}_0$ on the example document.

Connection to circuits. We remark that our mapping DAG can be seen as a kind of Boolean circuit, and our enumeration algorithm on mapping DAGs can be connected to earlier work by some of the present authors on enumeration for Boolean circuits [25, 24]. Specifically, a mapping DAG can be understood as describing a kind of binary decision diagram (BDD): these are special kind of Boolean circuits where each conjunction always involves a literal. This class is more restricted than the circuits obtained for tree automata in [25, 24], intuitively because trees feature branching which require the conjunction of multiple sub-runs. Our enumeration algorithm on mapping DAGs in

the present work could then be phrased as a generic algorithm on a class of bounded-width, nondeterministic BDDs. However, in this work, we chose to eschew the circuit terminology, as we believe that our definitions and algorithms are simpler to present on an ad-hoc mapping DAG data structure.

Trimming, levels, and level sets. Our task is to enumerate $\mathcal{M}(G)$ *without duplicates*, and this is still non-obvious: because of nondeterminism, the same mapping in the product DAG may be witnessed by exponentially many paths, corresponding to exponentially many runs of the nondeterministic extended VA $\mathcal{C}$. We will present in the next section our algorithm to perform this task on the product DAG G. To do this, we will need to preprocess G by *trimming* it, and introduce the notion of *levels* to reason about its structure.

First, we present how to *trim* G. We say that G is *trimmed* if every vertex v is both *accessible* (there is a path from the initial vertex to v) and *co-accessible* (there is a path from v to the final vertex). Given a mapping DAG, we can clearly trim in linear time by two linear-time graph traversals. Hence, we will always implicitly assume that the mapping DAG is trimmed. If the mapping DAG may be empty once trimmed, then there are no mappings to enumerate, so our task is trivial. Hence, we assume in the sequel that the mapping DAG is non-empty after trimming. Further, if $\mathcal{V} = \emptyset$ then the only possible mapping is the empty mapping and we can produce it at that stage, so in the sequel we assume that $\mathcal{V}$ is non-empty.

Example 9.10. *For the mapping DAG of Figure 9.1, trimming eliminates the non-accessible vertices (which are not depicted) and the non-co-accessible vertices (i.e., those with incoming dashed edges). Note that trimming the mapping DAG has an effect even though the example sequential extended VA $\mathcal{C}_0$ was already trimmed.*

Second, we present an invariant on the structure of G by introducing the notion of *levels*:

Definition 9.11. A mapping DAG G is *leveled* if its vertices $v = (q, i)$ are pairs whose second component i is a nonnegative integer called the *level* of the vertex and written $\mathsf{level}(v)$, and where the following conditions hold:

- For the initial vertex v_0 (which has no incoming edges), the level is 0;
- For every ϵ-edge from u to v, we have $\mathsf{level}(v) = \mathsf{level}(u) + 1$;
- For every marker edge from u to v, we have $\mathsf{level}(v) = \mathsf{level}(u)$. Furthermore, all pairs (m, i) in the label of the edge have $i = \mathsf{level}(v)$.

The *depth* D of G is the maximal level. The *width* W of G is the maximal number of vertices that have the same level.

The following is then immediate by construction:

Claim 9.12. *The product DAG of $\mathcal{C}$ and d is leveled, and we have $W \leq |Q|$ and $D = |d| + 1$.*

Proof. It is clear by construction that the product DAG satisfies the first three points in the definition of a leveled mapping DAG. To see why the last point holds, observe that for every edge of the product DAG, for every pair (m, i) that occurs in the label of that edge, the second component i of the pair indicates how many letters of d have been read so far, so the source vertex must have level i.

To see why the width and depth bounds hold, observe that each level of the product DAG corresponds to a copy of $\mathcal{C}$, so it has at most $|Q|$ vertices; and that the number of levels corresponds to the number of letters of the document, plus one level for the final vertex. □

Example 9.13. *The example mapping DAG on Figure 9.1 is leveled, and the levels are represented as horizontal layers separated by dotted lines: the topmost level is level 0 and the bottommost level is level 10.*

In addition to levels, we will need the notion of a *level set*:

Definition 9.14. A *level set* Λ is a non-empty set of vertices in a leveled normalized mapping DAG that all have the same level (written $\mathsf{level}(\Lambda)$) and which are all the source of some marker edge. The singleton $\{v_\mathrm{f}\}$ of the final vertex is also considered as a level set.

In particular, letting v_0 be the initial vertex, the singleton $\{v_0\}$ is a level set. Further, if we consider a level set Λ which is not the final vertex, then we can follow marker edges from all vertices of Λ (and only such edges) to get to other vertices, and follow ϵ-edges from these vertices (and only such edges) to get to a new level set Λ' with $\mathsf{level}(\Lambda') = \mathsf{level}(\Lambda) + 1$.

9.4. Enumeration for Mapping DAGs

In the previous section, we have reduced our enumeration problem for extended VAs on documents to an enumeration problem on normalized leveled mapping DAGs. In this section, we describe our main enumeration algorithm on such DAGs and show the following:

Theorem 9.15. *Let $2 \leq \omega \leq 3$ be an exponent for Boolean matrix multiplication. Given a normalized leveled mapping DAG G of depth D and width W, we can enumerate $\mathcal{M}(G)$ (without duplicates) with preprocessing $O(|G| + D \times W^{\omega+1})$ and delay $O(W^2 \times (r+1))$ where r is the size of each produced mapping.*

Remember that, as part of our preprocessing, we have ensured that the leveled normalized mapping DAG G has been trimmed. We will also preprocess G to ensure that, given any vertex, we can access its adjacency list (i.e., the list of its outgoing edges) in some sorted order on the labels, where we assume that $\emptyset$-edges come last. This sorting can be done in linear time on the RAM model [Gra96, Theorem 3.1], so the preprocessing is in $O(|G|)$.

Our general enumeration algorithm is then presented as Algorithm 1. We explain the missing pieces next. The function ENUM is initially called with $\Lambda = \{v_0\}$, the level set containing only the initial vertex, and with Mapping being the empty set.

Algorithm 1 Main enumeration algorithm

```
1: procedure ENUM(Λ, Mapping)
2:     Λ' := JUMP(Λ)
3:     if Λ' is the singleton {v_f} of the final vertex then
4:         OUTPUT(Mapping)
5:     else
6:         for (LocMark, Λ'') in NEXTLEVEL(Λ') do
7:             ENUM(Λ'', LocMark ∪ Mapping)
```

For simplicity, let us assume for now that the JUMP function just computes the identity, i.e., $\Lambda' := \Lambda$. As for the call NEXTLEVEL(Λ'), it returns the pairs $(\mathsf{LocMark}, \Lambda'')$ where:

- The label set LocMark is an edge label such that there is a marker edge e labeled with LocMark that starts at some vertex of Λ'
- The level set Λ'' is formed of all the vertices w at level $\mathsf{level}(\Lambda') + 1$ that can be reached by first following a marker edge e like in the bullet point above, and then following some ϵ-edge. Formally, a vertex w is in Λ'' if and only if there is an edge labeled LocMark from some vertex $v \in \Lambda'$ to some vertex v', and there is an ϵ-edge from v' to w.

Remember that, as the mapping DAG is normalized, we know that all edges starting at vertices of the level set Λ' are marker edges (several of which may have the same label); and for any target v' of these edges, all edges that leave v' are ϵ-edges whose targets w are at the level $\mathsf{level}(\Lambda') + 1$.

It is easy to see that the NEXTLEVEL function can be computed efficiently:

Proposition 9.16. *Given a leveled trimmed normalized mapping DAG G with width W, and a level set Λ', we can enumerate without duplicates all the pairs* $(\mathsf{LocMark}, \Lambda'') \in$ NEXTLEVEL(Λ') *with delay $O(W^2 \times |\mathsf{LocMark}|)$ in an order such that* $\mathsf{LocMark} = \emptyset$ *comes last if it is returned.*

Proof. The algorithm is outlined as Algorithm 2. Intuitively, we simultaneously go over the sorted lists of the outgoing edges of each vertex of Λ', of which there are at most W, and we merge them. Specifically, as long as we are not done traversing all lists, we consider the smallest value of LocMark (according to the order) that occurs at the current position of one of the lists. Then, we move forward in each list until the list is empty or the edge label at the current position is no longer equal to LocMark, and we consider the set Λ'_2 of all vertices v' that are the targets of the edges that we have seen. This considers at most W^2 edges and reaches at most W vertices (which are at the same level as Λ'), and the total time spent reading edge labels is in $O(|\mathsf{LocMark}|)$, so the process is in $O(W^2 \times |\mathsf{LocMark}|)$ so far. Now, we consider the outgoing edges of all vertices $v' \in \Lambda'_2$

Algorithm 2 Enumeration algorithm for Proposition 9.16

input: Level set $\Lambda' = \{v_1, \ldots, v_n\}$
for $j \in \{1, \ldots, n\}$ **do**
    $E_j \leftarrow$ outgoing edges of v_j
    $p_j \leftarrow 0$
while there is $1 \leq j \leq n$ such that $p_j < |E_j|$ **do**
    $\mathsf{LocMark} \leftarrow \min_{(j:p_j<|E_j|)} E_j[p_j].\text{label}$
    $\Lambda'_2 \leftarrow \emptyset$
    for $j \in \{1, \ldots, n\}$ **do**
        while $p_j < |E_j|$ and $E_j[p_j].\text{label} = \mathsf{LocMark}$ **do**
            $\Lambda'_2 \leftarrow \Lambda'_2 \cup \{E_j[p_j].\text{target}\}$
            $p_j \leftarrow p_j + 1$
    $\Lambda'' \leftarrow \emptyset$
    for $v' \in \Lambda'_2$ **do**
        for e outgoing edge of v' **do**
            $\Lambda'' \leftarrow \Lambda'' \cup \{e.\text{target}\}$
    OUTPUT($\mathsf{LocMark}, \Lambda''$)

(all are ϵ-edges) and return the set Λ'' of the vertices w to which they lead: this only adds $O(W^2)$ to the running time because we consider at most W vertices v' with at most W outgoing edges each. Last, $\mathsf{LocMark} = \emptyset$ comes last because of our assumption on the order of adjacency lists. □

The design of Algorithm 1 is justified by the fact that, for any level set Λ', the set $\mathcal{M}(\Lambda')$ can be partitioned based on the value of $\mathsf{LocMark}$. Formally:

Claim 9.17. *For any level set Λ of G which is not the final vertex, we have:*

$$\mathcal{M}(\Lambda) \quad = \bigcup_{(\mathsf{LocMark},\Lambda'') \in \textsc{NextLevel}(\Lambda)} \{\mathsf{LocMark} \cup \alpha \mid \alpha \in \mathcal{M}(\Lambda'')\} \,. \tag{9.1}$$

Furthermore, this union is disjoint, non-empty, and none of its terms is empty.

Proof. The definition of a level set and of a normalized mapping DAG ensures that we can decompose any path π from Λ to v_f as a marker edge e from Λ to some vertex v', an ϵ-edge from v' to some vertex w, and a path π' from w to v_f. Further, the set of such w is clearly a level set. Hence, the left-hand side of Equation (9.1) is included in the right-hand side. Conversely, given such v, v', w, and π', we can combine them into a path π, so the right-hand side is included in the left-hand side. This proves Equation (9.1).

We show that the union is disjoint. Recall that the definition of a leveled mapping DAG (Definition 9.11) implies that $\mathsf{LocMark}$ is a set of pairs whose second component is $\mathsf{level}(\Lambda)$, and that each mapping in $\mathcal{M}(\Lambda'')$ is a set of pairs whose second components are values strictly greater than $\mathsf{level}(\Lambda)$. Thus, each mapping in $\mathcal{M}(\Lambda)$ can only be obtained

for the value of $\mathsf{LocMark}$ which is equal to the subset of the pairs of the mapping whose second component is $\mathsf{level}(\Lambda)$.

We show that the union is non-empty. This is because Λ is non-empty and its vertices must be co-accessible so they must have some outgoing marker edge, which implies that $\textsc{NextLevel}(\Lambda)$ is non-empty.

We last show that none of the terms of the union is empty. This is because, for each $(\mathsf{LocMark}, \Lambda'') \in \textsc{NextLevel}(\Lambda)$, we know that Λ'' is non-empty because the mapping DAG is trimmed so all vertices are co-accessible. □

Thanks to this claim, we could easily prove by induction that Algorithm 1 correctly enumerates $\mathcal{M}(G)$ when $\textsc{Jump}$ is the identity function. However, this algorithm would not achieve the desired delay bounds: indeed, it may be the case that $\textsc{NextLevel}(\Lambda')$ only contains $\mathsf{LocMark} = \emptyset$, and then the recursive call to $\textsc{Enum}$ would not make progress in constructing the mapping, so the delay would not generally be linear in the size of the mapping. To avoid this issue, we use the $\textsc{Jump}$ function to directly "jump" to a place in the mapping DAG where we can read a label different from $\emptyset$. Let us first give the relevant definitions:

Definition 9.18. Given a level set Λ in a leveled mapping DAG G, the *jump level* $\mathsf{JL}(\Lambda)$ of Λ is the first level $j \geq \mathsf{level}(\Lambda)$ containing a vertex v' such that some $v \in \Lambda$ has a path to v' and such that v' is either the final vertex or has an outgoing edge with a label which is $\neq \epsilon$ and $\neq \emptyset$. In particular we have $\mathsf{JL}(\Lambda) = \mathsf{level}(\Lambda)$ if some vertex in Λ already has an outgoing edge with such a label, or if Λ is the singleton set containing only the final vertex.

The *jump set* of Λ is then $\textsc{Jump}(\Lambda) := \Lambda$ if $\mathsf{JL}(\Lambda) = \mathsf{level}(\Lambda)$, and otherwise $\textsc{Jump}(\Lambda)$ is formed of all vertices at level $\mathsf{JL}(\Lambda)$ to which some $v \in \Lambda$ have a directed path whose last edge is labeled ϵ. This ensures that $\textsc{Jump}(\Lambda)$ is always a level set.

Example 9.19. *In the mapping DAG in Figure 9.1, we have* $\mathsf{JL}(\{(q_2, 3), (q_5, 3)\}) = 5$*, as the reachable node* $(q_9, 5)$ *has an outgoing edge labeled* $\{(\dashv x, 5)\}$*. The jump set* $\textsc{Jump}(\{(q_2, 3), q_5, 3)\})$ *is* $\{(q_2, 5), (q_9, 5)\}$*, as* $(q_2, 5)$ *is reachable from* $(q_2, 3)$ *and* $(q_9, 5)$ *is reachable from* $(q_5, 3)$*.*

The definition of $\textsc{Jump}$ ensures that we can jump from Λ to $\textsc{Jump}(\Lambda)$ when enumerating mappings, and it will not change the result because we only jump over ϵ-edges and $\emptyset$-edges:

Claim 9.20. *For any level set* Λ *of* G*, we have* $\mathcal{M}(\Lambda) = \mathcal{M}(\textsc{Jump}(\Lambda))$*.*

Proof. As $\textsc{Jump}(\Lambda)$ contains all vertices from level $\mathsf{JL}(\Lambda)$ that can be reached from Λ, any path π from a vertex $u \in \Lambda$ to the final vertex can be decomposed into a path π_{uw} from u to a vertex $w \in \textsc{Jump}(\Lambda)$ and a path π_{wv} from w to v. By definition of $\textsc{Jump}(\Lambda)$, we know that all edges in π_{uw} are labeled with ϵ or $\emptyset$, so $\mu(\pi) = \mu(\pi_{wv})$. Hence, we have $\mathcal{M}(\Lambda) \subseteq \mathcal{M}(\textsc{Jump}(\Lambda))$.

Conversely, given a path π_{wv} from a vertex $w \in \textsc{Jump}(\Lambda)$ to the final vertex, the definition of $\textsc{Jump}(\Lambda)$ ensures that there is a vertex $u \in \Lambda$ and a path π_{uw} from u to w, which again consists only of ϵ-edges or $\emptyset$-edges. Hence, letting π be the concatenation of

π_{uw} and π_{wv}, we have $\mu(\pi_{wv}) = \mu(\pi)$ and π is a path from Λ to the final vertex. Thus, we have $\mathcal{M}(\textsc{Jump}(\Lambda)) \subseteq \mathcal{M}(\Lambda)$, concluding the proof. □

Claims 9.17 and 9.20 imply that Algorithm 1 is correct with this implementation of JUMP:

Proposition 9.21. $\textsc{Enum}(\{v_0\}, \epsilon)$ *correctly enumerates* $\mathcal{M}(G)$ *(without duplicates).*

Proof. We show the stronger claim that for every level set Λ, and for every sequence of labels $\mathsf{Mapping}$, we have that $\textsc{Enum}(\Lambda, \mathsf{Mapping})$ enumerates (without duplicates) the set $\mathsf{Mapping} \uplus \mathcal{M}(\Lambda) := \{\mathsf{Mapping} \cup \alpha \mid \alpha \in \mathcal{M}(\Lambda)\}$. The base case is when Λ is the final vertex, and then $\mathcal{M}(\Lambda) = \{\{\}\}$ and the algorithm correctly returns $\{\mathsf{Mapping}\}$.

For the induction case, let us consider a level set Λ which is not the final vertex, and some sequence of labels $\mathsf{Mapping}$. We let $\Lambda' := \textsc{Jump}(\Lambda)$, and by Claim 9.20 we have that $\mathcal{M}(\Lambda') = \mathcal{M}(\Lambda)$. Now we know by Claim 9.17 that $\mathcal{M}(\Lambda')$ can be written as in Equation (9.1) and that the union is disjoint; the algorithm evaluates this union. So it suffices to show that, for each $(\mathsf{LocMark}, \Lambda'') \in \textsc{NextLevel}(\Lambda')$, the corresponding iteration of the **for** loop enumerates (without duplicates) the set $(\mathsf{Mapping} \cup \mathsf{LocMark}) \uplus \mathcal{M}(\Lambda'')$. By induction hypothesis, the call $\textsc{Enum}(\textsc{Jump}(\Lambda'), \mathsf{Mapping} \cup \mathsf{LocMark})$ enumerates (without duplicates) the set $(\mathsf{Mapping} \cup \mathsf{LocMark}) \uplus \mathcal{M}(\textsc{Jump}(\Lambda''))$. So this establishes that the algorithm is correct. □

What is more, Algorithm 1 now achieves the desired delay bounds, as we will show. Of course, this relies on the fact that the JUMP function can be efficiently precomputed and evaluated. We only state this fact for now, and prove it in the next section:

Proposition 9.22. *Given a leveled mapping DAG* G *with width* W *and depth* D*, we can preprocess* G *in time* $O(D \times W^{\omega+1})$ *such that, given any level set* Λ *of* G*, we can compute the jump set* $\textsc{Jump}(\Lambda)$ *of* Λ *in time* $O(W^2)$*.*

We can now conclude the proof of Theorem 9.15 by showing that the preprocessing and delay bounds are as claimed. For the preprocessing, this is clear: we do the preprocessing in $O(|G|)$ presented at the beginning of the section (i.e., trimming, and computing the sorted adjacency lists), followed by that of Proposition 9.22. For the delay, we claim:

Claim 9.23. *Algorithm 1 has delay* $O(W^2 \times (r+1))$*, where* r *is the size of the mapping of each produced path. In particular, the delay is independent of the size of* G*.*

Proof. Let us first bound the delay to produce the first solution. When we enter the ENUM function, we call the JUMP function to produce Λ' in time $O(W^2)$ by Proposition 9.22, and either Λ' is the final vertex or some vertex in Λ' must have an outgoing edge with a label different from $\emptyset$. Then we enumerate $\textsc{NextLevel}(\Lambda')$ with delay $O(W^2 \times |\mathsf{LocMark}|)$ for each $\mathsf{LocMark}$ using Proposition 9.16. Remember that Proposition 9.16 ensures that the label $\emptyset$ comes last; so by definition of JUMP the first value of $\mathsf{LocMark}$ that we consider is different from $\emptyset$. At each round of the **for** loop, we recurse in constant time: in particular, we do not copy $\mathsf{Mapping}$ when writing $\mathsf{LocMark} \cup \mathsf{Mapping}$, as we can represent the set simply as a linked list. Eventually, after $r+1$ calls, by definition of a leveled mapping

DAG, Λ must be the final vertex, and then we output a mapping of size r in time $O(r)$: the delay is indeed in $O(W^2 \times (r+1))$ because the sizes of the values of $\mathsf{LocMark}$ seen along the path sum up to r, and the unions of $\mathsf{LocMark}$ and $\mathsf{Mapping}$ are always disjoint by definition of a mapping DAG.

Let us now bound the delay to produce the next solution. To do so, we will first observe that when enumerating a mapping of cardinality r, then the size of the recursion stack is always $\leq r+1$. This is because Proposition 9.16 ensures that the value $\mathsf{LocMark} = \emptyset$ is always considered last in the **for** loop on NextLevel(Λ'). Thanks to this, every call to Enum where $\mathsf{LocMark} = \emptyset$ is actually a tail recursion, and we can avoid putting another call frame on the call stack using tail recursion elimination. This ensures that each call frame on the stack (except possibly the last one) contributes to the size of the currently produced mapping, so that indeed when we reach the final vertex of G then the call stack is no greater than the size of the mapping that we produce.

Now, let us use this fact to bound the delay between consecutive solutions. When we move from one solution to another, it means that some **for** loop has moved to the next iteration somewhere in the call stack. To identify this, we must unwind the stack: when we produce a mapping of size r, we unwind the stack until we find the next **for** loop that can move forward. By our observation on the size of the stack, the unwinding takes time $O(r)$ with r is the size of the previously produced mapping; so we simply account for this unwinding time as part of the computation of the previous mapping. Now, to move to the next iteration of the **for** loop and do the computations inside the loop, we spend a delay $O(W^2 \times |\mathsf{LocMark}|)$ by Proposition 9.16. Let r' be the current size of $\mathsf{Mapping}$, including the current $\mathsf{LocMark}$. The **for** loop iteration finishes with a recursive call to Enum, and we can re-apply our argument about the first solution above to argue that this call identifies a mapping of some size r'' in delay $O(W^2 \times (r''+1))$. However, because the argument $\mathsf{Mapping}$ to the recursive call had size r', the mapping which is enumerated actually has size $r' + r''$ and it is produced in delay $O(W^2 \times (r''+1) + r')$. This means that the overall delay to produce the next solution is indeed in $O(W^2 \times (r+1))$ where r is the size of the mapping that is produced, which concludes the proof. □

Memory usage. We briefly discuss the *memory usage* of the enumeration phase, i.e., the maximal amount of working memory that we need to keep throughout the enumeration phase, not counting the precomputation phase. Indeed, in enumeration algorithms the memory usage can generally grow to be very large even if one adds only a constant amount of information at every step. We will show that this does not happen here, and that the memory usage throughout the enumeration remains polynomial in $\mathcal{C}$ and constant in the input document size.

All our memory usage during enumeration is in the call stack, and thanks to tail recursion elimination (see the proof of Claim 9.23) we know that the stack depth is at most $r+1$, where r is the size of the produced mapping as in the statement of Theorem 9.15. The local space in each stack frame must store Λ' and Λ'', which have size $O(W)$, and the status of the enumeration of NextLevel in Proposition 9.16, i.e., for every vertex $v \in \Lambda'$, the current position in its adjacency list: this also has total size $O(W)$, so the total memory usage of these structures over the whole stack is in

$O((r+1) \times W)$. Last, we must also store the variables Mapping and LocMark, but their total size of the variables LocMark across the stack is clearly r, and the same holds of Mapping because each occurrence is stored as a linked list (with a pointer to the previous stack frame). Hence, the total memory usage is $O((r+1) \times W)$, i.e., $O((|\mathcal{V}|+1) \times |Q|)$ in terms of the extended VA.

9.5. Jump Function

The only missing piece in the enumeration scheme of Section 9.4 is the proof of Proposition 9.22. We first explain the preprocessing for the JUMP function, and then the computation scheme.

Preprocessing scheme. Recall the definition of the jump level $\mathsf{JL}(\Lambda)$ and jump set $\textsc{Jump}(\Lambda)$ of a level set Λ (Definition 9.18). We assume that we have precomputed in $O(|G|)$ the mapping level associating each vertex v to its level $\mathsf{level}(v)$, as well as, for each level i, the list of the vertices v such that $\mathsf{level}(v) = i$.

The first part of the preprocessing is then to compute, for every individual vertex v, the jump level $\mathsf{JL}(v) := \mathsf{JL}(\{v\})$, i.e., the minimal level containing a vertex v' such that v' is reachable from v and v' is either the final vertex or has an outgoing edge which is neither an ϵ-edge nor an $\emptyset$-edge. We claim:

Claim 9.24. *We can precompute in $O(D \times W^2)$ the jump level $\mathsf{JL}(v)$ of all vertices v of G.*

Proof. This construction can be performed iteratively from the final vertex v_f to the initial vertex v_0: we have $\mathsf{JL}(v_\mathrm{f}) := \mathsf{level}(v_\mathrm{f})$ for the final vertex v_f, we have $\mathsf{JL}(v) := \mathsf{level}(v)$ if v has an outgoing edge which is not an ϵ-edge or an $\emptyset$-edge, and otherwise we have $\mathsf{JL}(v) := \min_{v \to w} \mathsf{JL}(w)$.

This computation can be performed along a reverse topological order, which by [CLRS09, Section 22.4] takes linear time in G. However, note that G has at most $D \times W$ vertices, and we only traverse ϵ-edges and $\emptyset$-edges: we just check the existence of edges with other labels but we do not traverse them. Now, as each vertex has at most W outgoing edges labeled $\emptyset$ and at most W outgoing edges labeled ϵ, the number of edges in the DAG that we actually traverse is only $O(D \times W^2)$, which shows our complexity bound and concludes the proof. □

The second part of the preprocessing is to compute, for each level i of G, the *reachable levels* $\mathsf{Rlevel}(i) := \{\mathsf{JL}(v) \mid \mathsf{level}(v) = i\}$, which we can clearly do in linear time in the number of vertices of G, i.e., in $O(D \times W)$. Note that the definition clearly ensures that we have $|\mathsf{Rlevel}(i)| \leq W$.

Example 9.25. *In Figure 9.1, the jumping level for nodes $(q_1, 3)$ and $(q_2, 3)$ is 6 and the jumping level for nodes $(q_5, 3)$ and $(q_6, 3)$ is 5. Hence, the set of reachable levels $\mathsf{Rlevel}(3)$ for level 3 is $\{5, 6\}$.*

Last, the third step of the preprocessing is to compute a reachability matrix from each level to its reachable levels. Specifically, for any two levels $i < j$ of G, let $\mathsf{Reach}(i, j)$ be the Boolean matrix of size at most $W \times W$ which describes, for each (u, v) with $\mathsf{level}(u) = i$ and $\mathsf{level}(v) = j$, whether there is a path from u to v whose last edge is labeled ϵ. We can't afford to compute all these matrices, but we claim that we can efficiently compute a subset of them, which will be enough for our purposes:

Claim 9.26. *We can precompute in time $O(D \times W^{\omega+1})$ the matrices $\mathsf{Reach}(i, j)$ for all pairs of levels $i < j$ such that $j \in \mathsf{Rlevel}(i)$.*

Proof. We compute the matrices in decreasing order on i, then for each fixed i in arbitrary order on j:

- if $j = i$, then $\mathsf{Reach}(i, j)$ is the identity matrix;
- if $j = i + 1$, then $\mathsf{Reach}(i, j)$ can be computed from the edge relation of G in time $O(W \times W)$, because it suffices to consider the edges labeled $\emptyset$ and ϵ between levels i and j;
- if $j > i + 1$, then $\mathsf{Reach}(i, j)$ is the product of $\mathsf{Reach}(i, i + 1)$ and $\mathsf{Reach}(i + 1, j)$, which can be computed in time $O(W^\omega)$.

In the last case, the crucial point is that $\mathsf{Reach}(i + 1, j)$ has already been precomputed, because we are computing Reach in decreasing order on i, and because we must have $j \in \mathsf{Rlevel}(i + 1)$. Indeed, if $j \in \mathsf{Rlevel}(i)$, then there is a vertex v with $\mathsf{level}(v) = i$ such that $\mathsf{JL}(v) = j$, and the inductive definition of JL implies that v has an edge to a vertex w such that $\mathsf{level}(w) = i + 1$ and $\mathsf{JL}(v) = \mathsf{JL}(w) = j$, which witnesses that $j \in \mathsf{Rlevel}(i + 1)$.

The total running time of this scheme is in $O(D \times W^{\omega+1})$: indeed we consider each of the D levels of G, we compute at most W matrices for each level of G because we have $|\mathsf{Rlevel}(i)| \leq W$ for any i, and each matrix is computed in time at most $O(W^\omega)$. □

Evaluation scheme. We can now describe our evaluation scheme for the jump function. Given a level set Λ, we wish to compute $\textsc{Jump}(\Lambda)$. Let i be the level of Λ, and let j be $\mathsf{JL}(\Lambda)$ which we compute as $\min_{v \in \Lambda} \mathsf{JL}(v)$ in $O(W)$ time. If $j = i$, then $\textsc{Jump}(\Lambda) = \Lambda$ and there is nothing to do. Otherwise, by definition there must be $v \in \Lambda$ such that $\mathsf{JL}(v) = j$, so v witnesses that $j \in \mathsf{Rlevel}(i)$, and we know that we have precomputed the matrix $\mathsf{Reach}(i, j)$. Now $\textsc{Jump}(\Lambda)$ are the vertices at level j to which the vertices of Λ (at level i) have a directed path whose last edge is labeled ϵ, which we can simply compute in time $O(W^2)$ by unioning the lines that correspond to the vertices of Λ in the matrix $\mathsf{Reach}(i, j)$.

This concludes the proof of Proposition 9.22 and completes the presentation of our scheme to enumerate the set captured by mapping DAGs (Theorem 9.15). Together with Section 9.3, this proves Theorem 9.3 in the case of extended sequential VAs.

9.6. Experimental Validation

Having concluded the proof of our main result, we move on in this section to an experimental study of a prototype implementation of our method. A first direct implementation of our algorithm was developed by Rémi Dupré during his master thesis, which we further optimized to achieve better results, in particular to improve the handling of the reachability matrices and the space usage. In this section, we present this optimized implementation and show how it performs on several benchmarks. Our software is written in Rust and is freely available online[1] under the BSD 3-clause license.

Overall design. Our implementation enumerates the solutions of the evaluation of a nondeterministic sequential VA over a word. The nondeterministic sequential VA is given in the input as a regex-formula. This regex-formula is translated into a nondeterministic sequential VA using a variant of Glushkov's algorithm. Note that our implementation uses variable-set automata so the underlying algorithm could work with any regular spanner, and not only with hierarchical regular spanners [FKRV15, Theorem 4.6]. As for the input document, it is provided as a text file.

Our implementation follows the different parts of the algorithm presented in the paper. The preprocessing phase comprises (i) the construction of the mapping DAG as described in Section 9.3 and modified for non-extended VAs as described in [20]; and (ii) the construction of the jump function described in Section 9.5 and all necessary matrices. The enumeration phase explores the DAG as described in Section 9.4 and modified for non-extended VAs in [20]. In particular, we use the flashlight search approach described in [20].

9.6.1. Optimizations

Our optimizations focus on three main problems: efficiently managing the mapping DAG during the preprocessing phase, managing the reachability matrices that we build at the end of the preprocessing phase, and optimizing the enumeration phase.

Efficient representation of the mapping DAG and efficient exploration. The first stage of the preprocessing phase is to compute the mapping DAG. This DAG is efficiently represented as a bitmap[2] in which we store which states are reachable at each position of the input document. To save space, the implementation does not actually store any edges of the DAG, as the edges can be reconstructed on the fly from the automaton and input string.

The second stage is to make this DAG trimmed by exploring it to remove the vertices that are not co-accessible, i.e., those that have no path to the final vertex.

[1] https://github.com/PoDMR/enum-spanner-rs

[2] The bitmap contains a single bit for each pair $(q, i) \in Q \times \{0, \ldots, |d|\}$ that says whether the node is part of the trimmed mapping DAG or not. Padding is applied to ensure that each level starts at a machine word boundary.

Implementation of the matrices. The third stage of the preprocessing is to compute the reachability matrices that are necessary for the jump function, which requires many Boolean matrix multiplications. We considered using optimized implementations of matrix multiplication, but these are generally designed for floating-point numbers rather than Boolean values, so using them would significantly increase the memory usage. As memory space tends out to be an important bottleneck in our implementation, we instead implemented our own matrix multiplication code: it uses the naive matrix multiplication algorithm with three nested loops, but we optimized it for Boolean matrices as follows. We store matrices as bitvectors and pad their width to 8, 16, 32, or a multiple of 64, which reduces their memory usage. Further, we use fast bitwise operations in the inner loop of the matrix multiplication algorithm, which speeds up the multiplication of large matrices by a factor of up to 64. With this vectorized implementation, the multiplication time grows roughly like n^2 for matrices with width up to 64.

Enumeration phase. After these optimizations to the three stages of the preprocessing phase, our implementation performs the enumeration phase by traversing the mapping DAG *in reverse*, i.e., we explore it backwards from the final vertex to the initial vertices. Following this reverse order, we then enumerate the mappings seen along the traversed paths as we previously described in the paper. One advantage of doing enumeration backwards is that it allows us to skip the trimming step (second stage of the preprocessing phase): if some vertices of the mapping DAG are not co-accessible, the enumeration phase will never reach them and the delay bounds are not affected. However, as we will later show, in practice the time spent on trimming (second preprocessing stage) is often recouped during the third preprocessing stage (because it runs faster when the mapping DAG is smaller).

A more distant benefit of processing the DAG backwards is to later extend our implementation to support *updates*, i.e., modifications to the underlying document. A common case of updates is appending characters at the end, which we believe would be easier to handle when enumeration starts at the end. Nevertheless, the question of extending the algorithm and implementation to handle updates is left for future work (see also the discussion in the conclusion).

9.6.2. Experiments

Experimental setup and delay measurement. The tests were run in a virtual machine that had exclusive access to two Xeon E5-2630 CPU cores. The algorithm is single-threaded, but the additional core was added to minimize the effects of background activity of the operating system.

Measuring the delays of the algorithm is challenging, because the timescale for the delays is so tiny that unavoidable hardware interrupts can make a big difference. To eliminate outliers resulting from such interrupts, we exploited the fact that our enumeration algorithm is fully deterministic. We ran the algorithm ten times and recorded all delays. Afterwards, for each produced result, we took the median of the ten delays we collected. All measurements related to delays use this approach, e.g., if we compute the maximum

delay for a query, it is actually the maximum over these medians.

We benchmarked our implementation on two data sets: one based on genetic data and another one based on blog posts using the corpus from [SKAP06] and comparing against [Mor17]. We first describe the experiments on DNA data, and then the experiments on blog posts.

DNA data. For our experiments on DNA data, the input document is the first chromosome of the human genome reference sequence GRCh38. It contains roughly 250 million base pairs[3], where each base pair is encoded as a single character. We also use prefixes of this data in the experiments, when we need to benchmark against input documents of various sizes.

In most queries, there are no named capture variables, but there is an implicit capture variable which captures each possible match of the regex as a subword of the input document. Formally, when we write a query in the sequel as a regular expression e without capture variables, the corresponding spanner is the one described by the regex-formula $\Sigma^* x\{e\}\Sigma^*$, where Σ is the alphabet and x is the implicit capture variable.

Close-fragments queries. Our experiments on DNA data use so-called *close-fragment queries*, where we search for two DNA fragments w_1 and w_2 that occur close to each other. Specifically, we used the query `TTAC.{0,k}CACC`, with various values of k, for several different tests which we list below and then present in more detail.

1. We first verified that the delay is independent from the document length, while the preprocessing time and memory usage depends linearly on the document length. This is presented in Figure 9.2.

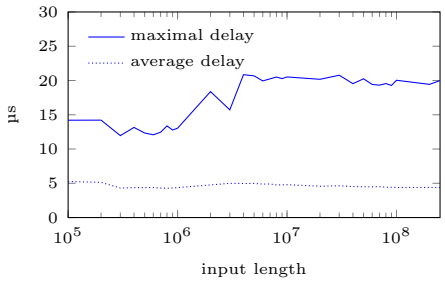
2. We then tested how the preprocessing time, the index structure size and the delay depends on the automaton size. This is depicted in Figure 9.3, where we used a 10 MB prefix of the DNA string and used values of k between 10 and 10 000.

3. Last, we compared the total enumeration time with the naive approach that starts one run of the NFA at every position of the document.[4] We also investigated the effect of skipping the second stage of the preprocessing. The results are depicted in Figure 9.4.

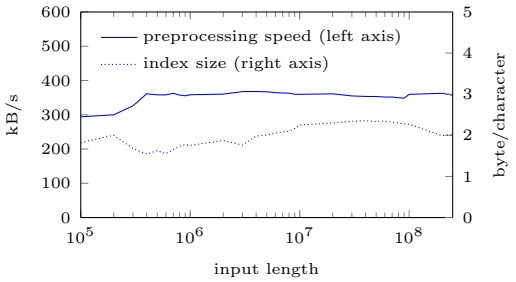
For (1), we fixed $k = 1\,000$ and used prefixes of different length of the DNA string. The results are depicted in Figure 9.2, where in Figure 9.2a, we depicted the maximal and average delay encountered during enumeration, while in Figure 9.2b, we depicted the preprocessing time and size of the index structure divided by the input length. One can see that the average delay is constant (around five microseconds allowing to enumerate 200 000 results per second), while the maximum delay is roughly four times larger. The

[3] https://www.ncbi.nlm.nih.gov/genome/guide/human/

[4] Note that this naive approach only works for the special case where there is exactly one capture variable that surrounds the whole expression. Our implementation has the added advantage of handling regular spanners with arbitrarily many capture variables.

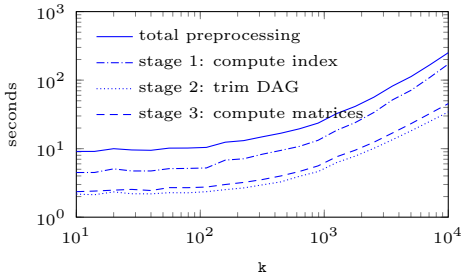


(a) Enumeration delay

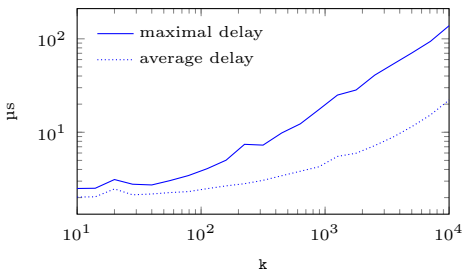


(b) Preprocessing speed and index structure size

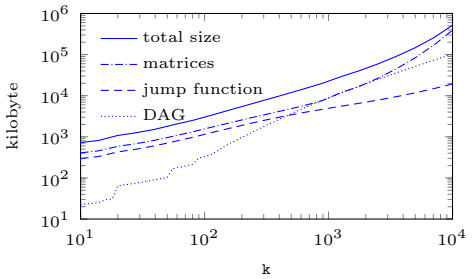
Figure 9.2.: Enumerating the query `TTAC.{0,1000}CACC` on inputs of different lengths



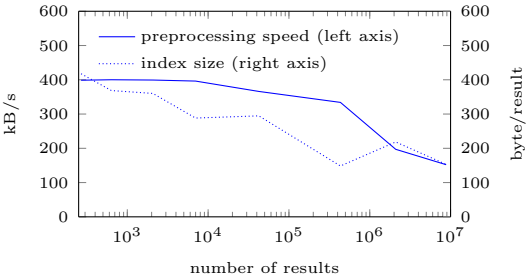
(a) Preprocessing time



(b) Enumeration delay

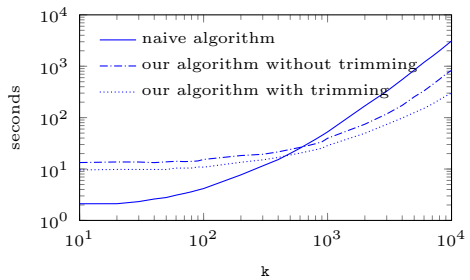


(c) Size of index structure

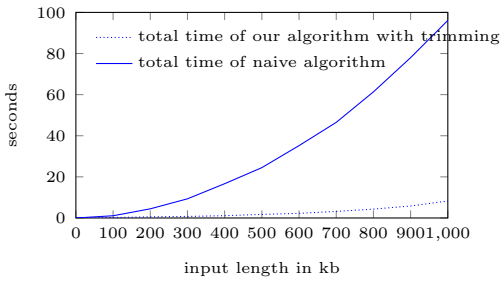


(d) Enumerating queries $w_1.\{0,1000\}w_2$

Figure 9.3.: Enumerating the query `TTAC.{0,k}CACC` on an input document of 10MB



(a) `TTAC.{0,k}CACC` over 10 MB



(b) `TTAC.*CACC`

Figure 9.4.: Comparing the total enumeration time with a simple algorithm

preprocessing speed is roughly 300 to 350 kilobytes per second and the index structure twice as large as the input document.

Towards (2), we fixed the length of the input to 10 MB and made k vary between 10 and 10 000. The results are shown in Figures 9.3a and 9.3b. The most interesting outcome is that the preprocessing is much faster than the worst case bound of $O(k^4)$. Analyzing the numbers from Figure 9.3a shows that the preprocessing time grows roughly like $\Theta(k^2)$. A closer look into the index structure used in the algorithm suggests an explanation: the width of the mapping DAG seems to grow sublinearly as a function of k for this query.

As for the delay in Figure 9.3b, remember that our experiment is about changing the query, so the delay bound is not supposed to be constant with respect k. The theoretical bounds suggest that the delay should be $O(k^2)$, which matches what we obtain experimentally for the maximum delay. The average delay is much lower.

We also measured the size of our index structure for the queries `TTAC.{0,k}CACC` after completing the preprocessing and depicted the results in Figure 9.3c. The index structure consists of three parts, with the total size being the sum of these three parts:

- DAG: The bitmap storing the states that exist in the (trimmed) DAG. We remove the levels to which the algorithm will never jump.
- Jump function: The jump function, as explained in Section 9.5.
- Matrices: All necessary reachability matrices, as explained in the same section.

For small automata, the size is dominated by the administrative overhead of the vectors used to store the jump functions and matrices, while the DAG is represented in a very compact way as a bitmap. For larger automata, one can see that the DAG representation uses more space, but the memory footprint is still dominated by the matrices. Notice that the size of each level of the DAG is padded to a multiple of 32, hence the bumps of the DAG curve around the sizes 32 and 64.

A question related to the close-fragment queries `TTAC.{0,k}CACC` is to understand if the change in performance across different values of k is only caused by the change in the number of results. To experiment with this, we fixed $k = 1000$ and benchmarked queries w_1`.{0,1 000}`w_2, where w_1 was a prefix of `TTACGG` and w_2 was a prefix of `CACCTG`, so as to make the number of results vary without changing the size of the automaton too much. The results are depicted in Figure 9.3d. The resulting index structure size for these queries indeed depends a lot on the number of results. This is expected as the index structure only contains information for levels that are used as the boundary of at least one span in the results. Specifically, the size grows slightly sublinearly. The preprocessing speed (and thus the preprocessing time) is almost constant until the number of results becomes sufficiently large to be comparable to the input size. This is because, before that point, the dominating term in the preprocessing time is the processing of the input and not the computations performed on the DAG.

For (3), we implemented a naive enumeration algorithm that works without any preprocessing, to serve as a baseline. It evaluates the NFA starting from each position

i in the input document and outputs a pair (i, j) for each position j where the NFA reaches an accepting state using the standard algorithm that computes for each position the set of possible states. We do the easy optimization of stopping the run for a starting position i if we reach an ending position j with no more reachable states. We depicted the total time used for enumeration of our approach and the naive algorithm in Figure 9.4, where we ran the query `TTAC.{0,k}CACC` for various sizes of k on the 10MB prefix of the DNA sequence (Figure 9.4a) and additionally the query `TTAC.*CACC` for various prefixes of the input DNA sequence (Figure 9.4b). For small k in Figure 9.4a, the naive algorithm has a clear advantage, as it does not need to compute any index structure. Also, for these queries the runtime is bounded by $O(nk)$, as all runs of the NFA have a length bounded by at most $k + 8$ because we optimized the baseline algorithms to stop the run early. For larger k, the naive algorithm is much slower than our approach. For the query `TTAC.*CACC` in Figure 9.4b, the naive approach exhibits its $\Theta(n^2)$ worst-case behavior, and is much slower than our approach, even for small input documents.

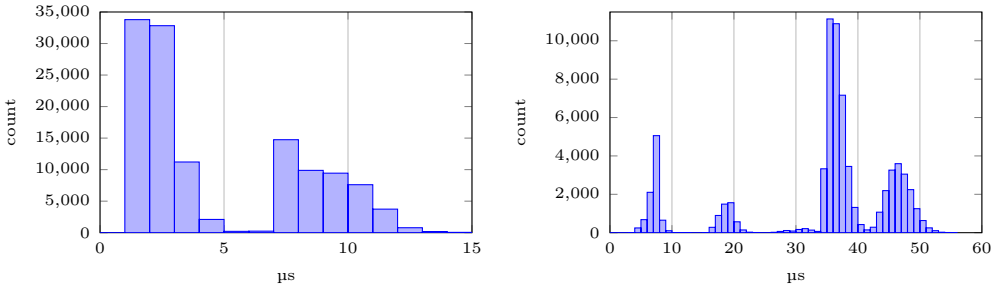
In Figure 9.4a, we also have a look on the effect of trimming the DAG (second stage of the preprocessing). Indeed, while skipping this trimming stage saves a small amount of time, this is usually overcompensated by the third preprocessing stage, where we need to compute more and larger matrices because the unpruned DAG is larger. This can be seen for the query `TTAC.{0,k}CACC` even for small values for k. Trimming saves more time for larger values of k, as more nodes of the DAG can be pruned. For the query `TTAC.*CACC` in Figure 9.4b, where trimming can only remove a few nodes from the DAG, the runtime effect of disabling trimming was negligible, i.e., the time savings from the second stage where almost exactly compensated by the additional work in the third stage.

Querying blog posts. We also evaluated our algorithm on roughly 800 megabytes of blog posts using the corpus from [SKAP06]. To apply our implementation, we concatenated all blog posts to get a single file and stripped all characters that did not have a valid UTF-8 encoding. We ran the same queries used in the master thesis of Morciano [Mor17, Chapter 6]. These queries try to extract reviews for movies from blog posts. They are built over simple dictionaries that contain, e.g., synonyms for "movie", synonyms for "actor", or a list of genres. These basic spanners are combined to more complex queries using the union operator and joins of the following form: "spanner B matches at most k characters after spanner A matches". For instance, the queries Q_1 to Q_4 are of the form: find a word in the dictionary d_1, and then a word in the dictionary d_2 matching at most k characters after the first word.

In Table 9.1, we give some statistical data over these queries, and give the running time of our algorithm, its memory usage, and the approximate times of the implementation of [Mor17]. We only report the time for the preprocessing phase of our algorithm, because the time taken by the enumeration phase is always less than one second. We stress that the running times of [Mor17] and our running times are not comparable, because the experimental setup is very different, the hardware in use is not the same, and the algorithm of [Mor17] is not an enumeration algorithm but simply produces all results. The point of our comparison is not to claim an improvement in running times relative to [Mor17], but to show that, on this existing dataset, the total running time of our

Query	#states	#variables	#results	preprocess (s)	memory (MB)	time of [Mor17](s)
Q_1	40	2	4 975	772	2.72	$\approx$ 780
Q_2	211	2	6 099	1 057	3.70	$\approx$ 1 100
Q_3	246	2	5 915	1 090	3.63	$\approx$ 1 200
Q_4	52	2	2 232	771	1.22	$\approx$ 810
Q_5	343	6	12 020	1 254	8.04	$\approx$ 2 780
Q'_6	661	8	19 561	1 704	16.00	$\approx$ 4 330
Q'_7	805	10	62 103	1 948	53.36	$\approx$ 5 100
Q'_8	813	10	70 509	1 956	60.02	$\approx$ 6 000

Table 9.1.: Querying blog data



(a) Query `TTAC.{0,1000}CACC` (b) Query Q'_8 from [Mor17]

Figure 9.5.: Histogram for delays between two outputs

approach is comparable to that of their implementation.

Looking into our running times, we notice that the dependency of the preprocessing time on the automaton size is again much less than the $O(|\mathcal{C}|^4)$ worst-case bound. Again, this is probably because the matches are sparse, i.e., there are only very few nodes per level and therefore the matrices are of almost constant size. Similarly to our experiments on DNA data, the preprocessing time and index structure size show a dependency on the number of matches, as we need to compute matrices for all levels where a variable is opened or closed for some match. Of course, as our preprocessing is linear in the input document, this dependency can only hold when the number of results is at most linear in the document.

Detailed analysis of delay. We did a more detailed analysis of the various delays that we obtain while running the enumeration phase of our algorithm. We show a histogram of the delays for the query `TTAC.{0,1000}CACC` on DNA data in Figure 9.5 (a), and for the query Q'_8 from [Mor17] on the blog post corpus in Figure 9.5 (b).

One can see that the delay varies, which is expected: our algorithm is constant-delay in the sense of enforcing a constant upper bound on the delay, but the effective delay

can vary from one output to the next. Specifically, the number of jumps that need to be performed between two outputs can be any number between one and the maximal number of variable markers encountered in a single match. Also the time needed for the flashlight search can vary within given limits.

In Figure 9.5 (a), as the DNA query has only one implicit capture variable covering the whole match and thus two variable markers, we have two spikes in the histogram. The first spike corresponds to the case where the next matching is found by just changing the end marker, while the second spike corresponds to the case where the two markers are changed, so that the flashlight search and jump functions have to be executed twice. In Figure 9.5 (b), as the query Q'_8 has ten variables, we notice that the maximal delay is larger and there are more spikes in the histogram.

9.7. Conclusion

We have shown that we can efficiently enumerate the mappings of sequential variable-set automata on input documents, achieving linear-time preprocessing and constant-delay in data complexity, while ensuring that preprocessing and delay are polynomial in the input VA even if it is not deterministic. This result was previously considered as unlikely by [FRU$^+$18], and it improves on the algorithms in [FKP18]: with our algorithm, the delay between outputs does not depend on the input document, whereas it had a linear dependency on the size of the input document in [FKP18].

In Section 9.6, we did a thorough practical evaluation of our approach. The most encouraging result is, that for several classes of queries, the algorithm runs much faster than the theoretical worst case analysis would suggest. An interesting open question raised by the experimental validation is whether it is possible to adapt our algorithm to NFAs with counters. We believe that queries that use a join condition of the form pattern A should be matched near pattern B are important in practice. These kind of queries intrinsically depend on the use of counters. As the efficiency of our algorithm crucially depends on the size of the underlying automata, a more efficient representation of counters that does not depend on encoding the counter value in the state of the automaton could allow for big improvements in the runtime.

We will consider different directions for future works. A first question is how to cope with changes to the input document without recomputing our enumeration index structure from scratch. This question has been recently studied for other enumeration algorithms, see e.g. [24, BKS17a, BKS17b, BKS18, LM14, Nie18, NS18], but for atomic update operations: insertion, deletion, and relabelings of single nodes. However, as spanners operate on text, we would like to use bulk update operations that modify large parts of the text at once: cut and paste operations, splitting or joining strings, or appending at the end of a file and removing from the beginning, e.g., in the case of log files with rotation. It may be possible to show better bounds for these operations than the ones obtained by modifying each individual letter [NS18, LM14], and we believe our implementation could be modified to do so, at least when appending new content at the end of the document.

Part III.

Appendix

A. Curriculum Vitae

Personal Information

Address	UFR des Sciences Jean Perrin Rue Jean Souvraz SP 18 62307 Lens Cedex France
E-Mail	mengel@cril.fr
Birth name	Stefan Senitsch

Research Area

intersection of computational complexity theory, algorithms and combinatorics; applications in database theory and artificial intelligence

Employment

Scientific Employment and Education

10/2015–	**Research scientist (chargé de recherche)**, Centre National de la Recherche Scientifique (CNRS), Centre de Recherche en Informatique de Lens (CRIL), Lens, France
10/2015–08/2021	**Adjunct assistant professor (chargé d'enseignement)**, Département d'Informatique de l'École polytechnique, Palaiseau, France
10/2013–09/2015	**Lix-Qualcomm postdoctoral fellow**, Équipe Algorithms and Complexity, LIX, École polytechnique, France
07/2013–09/2013	**Researcher** in the group of Peter Bürgisser at the Technische Universität Berlin, Germany
02/2009–07/2013	**PhD student**, University of Paderborn, Germany, Institute of Mathematics, working group Algebraic Complexity and Algorithmic Algebra,

supervisor:	Peter Bürgisser
thesis title:	*Conjunctive Queries, Arithmetic Circuits and Counting Complexity*
grade:	summa cum laude (with highest honors)
reviewers:	Peter Bürgisser, Friedhelm Meyer of der Heide, Luc Segoufin
jury members:	Peter Bürgisser, Hans Kleine Büning, Thorsten Lagemann, Friedhelm Meyer of der Heide, Eckhard Steffen

10/2001–03/2007 **Diploma in Mathematics**, Ilmenau University of Technology, with distinction, Grade: 1,0 (on a scale from 1,0 to 5,0)
Focus on discrete mathematics and theoretical computer science, Diploma thesis in complexity theory: *Ein kombinatorischer Beweis für das PCP-Theorem*, supervisor: Martin Dietzfelbinger

Non-Scientific Employment

10/2007–12/2008 **software developer**, ASCAD GmbH, Bochum, Germany, industry position, development mainly in Java, Lisp

04/2007–09/2007 **software developer**, Informations Design AG, Frankfurt, Germany, industry internship, development mainly in C++

Publications

In all references below, the authors list is sorted alphabetically. In particular, no information on the contribution of the different authors can be inferred from them.

Journal Publications

[1] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-Delay Enumeration for Nondeterministic Document Spanners. *ACM Trans. Database Syst.*, 46(1): 2:1–2:30, 2021. Revised and extended version of [20].

[2] Stefan Mengel and Sebastian Skritek. Characterizing Tractability of Simple Well-Designed Pattern Trees with Projection. *Theory Comput. Syst.*, 65(1): 3–41, 2021. Revised and extended version of [21].

[3] Stefan Mengel and Romain Wallon. Graph Width Measures for CNF-Encodings with Auxiliary Variables. *Journal of Artificial Intelligence Research*, 67: 409–436, 2020. Revised and extended version of [18].

[4] Mike Behrisch, Miki Hermann, Stefan Mengel, and Gernot Salzer. Minimal Distance of Propositional Models. *Theory of Computing Systems*, 63(6): 1131–1184, 2019. Revised and extended version of [32, 31, 30].

[5] Christian Ikenmeyer and Stefan Mengel. On the relative power of reduction notions in arithmetic circuit complexity. *Inf. Process. Lett.*, 130:7–10, 2018.

[6] Stefan Mengel. Lower Bounds on the mim-width of Some Graph Classes. *Discrete Applied Mathematics*, 248:28–32, 2018.

[7] Florent Capelli, Arnaud Durand, and Stefan Mengel. The arithmetic complexity of tensor contraction. *Theory Comput. Syst.*, 58(4):506–527, 2016. Revised and extended version of [39].

[8] Hervé Fournier, Guillaume Malod, and Stefan Mengel. Monomials in Arithmetic Circuits: Complete Problems in the Counting Hierarchy. *Computational Complexity*, 24(1):1–30, 2015. Revised and extended version of [40].

[9] Arnaud Durand and Stefan Mengel. Structural tractability of counting of solutions to conjunctive queries. *Theory Comput. Syst.*, 57(4):1202–1249, 2015. Revised and extended version of [38].

[10] Arnaud Durand and Stefan Mengel. The complexity of weighted counting for acyclic conjunctive queries. *J. Comput. Syst. Sci.*, 80(1):277–296, 2014.

[11] Jochen Harant and Stefan Senitsch. A generalization of Tutte's theorem on Hamiltonian cycles in planar graphs. *Discrete Mathematics*, 309(15):4949 – 4951, 2009.

Conference Proceedings

[12] Alexis de Colnet and Stefan Mengel. A Partial Succinctness Map for Positive Arithmetic Circuits. accepted at *18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, 2021

[13] Alexis de Colnet and Stefan Mengel. Characterizing Tseitin-formulas with short regular resolution refutations. *24th International Conference on Theory and Applications of Satisfiability Testing (SAT 2021)*, pages 116–133, 2021

[14] Stefan Mengel and Friedrich Slivovsky. Proof Complexity of Symbolic QBF Reasoning. *24th International Conference on Theory and Applications of Satisfiability Testing (SAT 2021)*, 2021

[15] Alexis de Colnet and Stefan Mengel. Lower Bounds for Approximate Knowledge Compilation. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI 2020)*, 2020.

[16] Daniel Le Berre, Pierre Marquis, Stefan Mengel, Romain Wallon. On Irrelevant Literals in Pseudo-Boolean Constraint Learning *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI 2020)*, 2020.

[17] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, Matthias Niewerth. Enumeration on Trees with Tractable Combined Complexity and Efficient Updates. *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2019)*, 2019.

[18] Stefan Mengel, Romain Wallon. Revisiting Graph Width Measures for CNF-Encodings *22nd International Conference on Theory and Applications of Satisfiability Testing (SAT 2019)*, 2019.

[19] Florent Capelli and Stefan Mengel. Tractable QBF by Knowledge Compilation. *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, 2019.

[20] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, Matthias Niewerth. Constant-Delay Enumeration for Nondeterministic Document Spanners. *22th International Conference on Database Theory (ICDT 2019)*, 2019.

[21] Stefan Mengel, Sebastian Skritek. Characterizing tractability of simple well-designed pattern trees with projection. *22th International Conference on Database Theory (ICDT 2019)*, 2019.

[22] Michael Lampis, Stefan Mengel, and Valia Mitsou. QBF as an alternative to courcelle's theorem. *21st International Conference on Theory and Applications of Satisfiability Testing (SAT 2018)*, pages 235–252, 2018.

[23] Daniel Le Berre, Pierre Marquis, Stefan Mengel, Romain Wallon. Pseudo-Boolean Constraints from a Knowledge Representation Perspective. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 1891 – 1897, 2018.

[24] Antoine Amarilli, Pierre Bourhis, and Stefan Mengel. Enumeration on trees under relabelings. *21st International Conference on Database Theory (ICDT 2018)*, pages 5:1–5:18, 2018.

[25] Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, pages 111:1–111:15, 2017.

[26] Hubie Chen and Stefan Mengel. The logic of counting query answers. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2017)*, pages 1 – 12, 2017.

[27] Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Knowledge Compilation Meets Communication Complexity. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 1008–1014, 2016.

[28] Hubie Chen and Stefan Mengel. Counting Answers to Existential Positive Queries: A Complexity Classification. *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2016)*, pages 315–326, 2016.

[29] Stefan Mengel. Parameterized Compilation Lower Bounds for Restricted CNF-Formulas. *19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016)*, pages 3–12, 2016.

[30] Mike Behrisch, Miki Hermann, Stefan Mengel, and Gernot Salzer. The Next Whisky Bar. *11th International Computer Science Symposium in Russia, (CSR 2016)*, pages 41–56, 2016.

[31] Mike Behrisch, Miki Hermann, Stefan Mengel, and Gernot Salzer. As Close as It Gets. *Algorithms and Computation - 10th International Workshop (WALCOM 2016)*, pages 222–235, 2016.

[32] Mike Behrisch, Miki Hermann, Stefan Mengel, and Gernot Salzer. Give Me Another One! *Algorithms and Computation - 26th International Symposium, (ISAAC 2015)*, pages 664–676, 2015.

[33] Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. On Compiling CNFs into Structured Deterministic DNNFs. *18th International Conference on Theory and Applications of Satisfiability Testing (SAT 2015)*, pages 199–214, 2015.

[34] Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for β-acyclic CNF-formulas. In *Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, pages 143–156, 2015.

[35] Hubie Chen and Stefan Mengel. A Trichotomy in the Complexity of Counting Answers to Conjunctive Queries. In *18th International Conference on Database Theory (ICDT 2015)*, pages 110–126, 2015.

[36] Florent Capelli, Arnaud Durand, and Stefan Mengel. Hypergraph acyclicity and propositional model counting. In *International Conference on Theory and Applications of Satisfiability Testing (SAT 2014)*, pages 399–414, 2014.

[37] Stefan Mengel. Arithmetic branching programs with memory. In *International Symposium on Mathematical Foundations of Computer Science (MFCS 2013)*, pages 667–678, 2013.

[38] Arnaud Durand and Stefan Mengel. Structural tractability of counting of solutions to conjunctive queries. In *International Conference on Database Theory (ICDT 2013)*, pages 81–92, 2013.

[39] Florent Capelli, Arnaud Durand, and Stefan Mengel. The arithmetic complexity of tensor contractions. In *Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, pages 365–376, 2013.

[40] Hervé Fournier, Guillaume Malod, and Stefan Mengel. Monomials in arithmetic circuits: Complete problems in the counting hierarchy. In *Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, pages 362–373, 2012.

[41] Stefan Mengel. Characterizing arithmetic circuit classes by constraint satisfaction problems - (extended abstract). In *International Colloquium on Automata, Languages and Programming (ICALP 2011)*, pages 700–711, 2011.

Preprints and Submitted

[42] Stefan Mengel. No Efficient Disjunction or Conjunction of Switch-Lists. 2020, submitted.

[43] Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. A strongly exponential separation of DNNFs from CNF formulas. *CoRR*, abs/1411.1995, 2014.

Teaching and Supervision

Student Supervision

PhD students

Alexis de Colnet, *Compilation de connaissances paramétrée et à grain fin* (co-supervised with Pierre Marquis, since 2019), joint publications: [15, 13, 12], best student paper award at SAT 2021 for [13]

Romain Wallon, *Raisonnement à partir de contraintes pseudo-booléennes et compilation* (co-supervised with Pierre Marquis and Daniel Le Berre, 2017-2020), joint publications: [23, 16, 18, 3], Honorable mention for the 2021 Thesis in AI award of the French Association for Artificial Intelligence (AFIA)

Master students

Baptiste Lafosse, *Solving the Cluster Editing problem*, internship Master 2, (co-supervised with Jean-Marie Lagniez, 2021)

Anthony Blomme, *L'abduction paramétrée*, internship Master 1, (co-supervised with Pierre Marquis, 2019), now PhD-student at CRIL

Romain Wallon, *Heuristiques pour la Décomposition : Application à la Compilation*, internship Master 1, (co-supervised with Pierre Marquis and Jean-Marie Lagniez, 2016), later my PhD student

PhD committees

reviewer and committee member for the PhD of Miloš Chromý, Charles University, Prague, title *"Boolean techniques in Knowledge representation"*, 2020

committee member for the PhD of Romain Wallon, Université d'Artois, Lens, title *Raisonnement à partir de contraintes pseudo-booléennes et compilation*, 2020

Teaching Experience

2020–2021 **CSE 304 - Complexity** tutorial classes, École Polytechnique, Palaiseau, France (third year bachelor students)

2017–2020 **CSE 101 - Computer Programming** tutorial classes, École Polytechnique, Palaiseau, France (first year bachelor students)

2016–2017 **INF442 - Traitement des données massives** tutorial classes, École Polytechnique, Palaiseau, France (second year Polytechnique students)

2015–2017 **INF311 - Introduction à l'informatique** tutorial classes, École Polytechnique, Palaiseau, France (first year Polytechnique students)

2012 **Combinatorics**, tutorials and stand-in lecturer for the lecture of Prof. Bürgisser, University of Paderborn (third year math students)

2011 **Introduction to Computer Algebra**, tutorials and stand-in lecturer for the lecture of Prof. Bürgisser, University of Paderborn (second year math students)

2011 **Linear Algebra for Computer Scientists**, help in organization of the tutorial, creation of exercises and solutions, stand-in lecturer for the lecture of Prof. Bürgisser, University of Paderborn (first year computer science students)

2004–2006 **Analysis III-IV, Functional Analysis**, tutorials for the lectures of Prof. Marx, Ilmenau University of Technology (second and third year math students)

Projects

EQUUS - Efficient Query answering Under UpdateS, ANR PRCI/DFG french-german project, project leader, 2020-2023

PING/ACK - Preprocessing Information for Nontrivial Goals / Advanced Compilation of Knowledge, ANR project, work package leader, 2019-2023

COMMODE - Knowledge COMpilation for feature MODEls, CPER project, region Hauts-de-France, member, 2019–2021

CODA - Compiling Provenance Data, PEPS JCJC INS2I (with Pierre Bourhis, CNRS CRIStAL, Lille), project leader, 2017

KOCOON - KnOwledge COmpilatiOn Network, in the program *Soutien à l'Animation de Collectifs de Recherche* of Hauts-de-France, member, 2019–2021

Reviewing and Administrative Roles

Program Committee Member for

Symposium on Principles of Database Systems (PODS): 2022

International Joint Conference on Artificial Intelligence (IJCAI): 2016–2021 (senior PC in 2021), Program Committee Board 2022–2024

AAAI Conference on Artificial Intelligence (AAAI): 2018, 2020, 2021, 2022

International Conference on Theory and Applications of Satisfiability Testing (SAT): 2016, 2018, 2021

International Conference on Database Theory (ICDT): 2021

International Symposium on Mathematical Foundations of Computer Science (MFCS): 2020

ACM/IEEE Symposium on Logic in Computer Science (LICS): 2019

International Symposium on Theoretical Aspects of Computer Science (STACS): 2018

Reviewer for

Conferences	WADS 2009, STACS 2011, FSTTCS 2011, STACS 2013, CP 2014, SOFSEM 2014, STACS 2015, TAMC 2015, CP 2015, ESA 2015, ICDT 2016, STACS 2016, PODS 2016, STACS 2017, AAAI 2017, CSR 2017, LICS 2017, ICALP 2017, ESA 2017, PODS 2018, CCC 2018, MFCS 2018, STACS 2019, FOCS 2019, ICDT 2020, SAT 2020, SODA 2020, FSTTCS 2020, ICALP 2021, CONCUR 2021
Journals	Algorithmica, Computational Complexity, Constraints, Information Processing Letters, Journal of Artificial Intelligence Research, Journal of the ACM, Logical Methods in Computer Science, ACM Transactions on Algorithms, Theoretical Computer Science

Workshop Organization

Workshop on Model Counting, colocated with SAT 2020, online

International Workshop on Graphs and Constraints, colocated with CP 2018, Lille, France

International Workshop on Graph Structure and Satisfiability Testing, colocated with SAT 2016, Bordeaux, France

Research Administration

treasurer of the SAT Association, the association in charge of the yearly SAT conference, by office also member of the board and the steering committee (since 2020)

member of the CRIL lab council (since 2020)

member of the hiring committee for a *maitre de conference* position at CRIL, 2017, 2021

responsible for the German translation of the CRIL websites (since 2018)

former co-responsible for the working group IMIA of GdR IM and GdR IA (2018-2020)

former organizer of the seminar of the AlCo group at LIX, Ecole Polytechnique

former webmaster for the Séminaire algorithmique du plateau de Saclay

References

[AAC+19] S. Akshay, Jatin Arora, Supratik Chakraborty, Shankara Narayanan Krishna, Divya Raghunathan, and Shetal Shah. Knowledge compilation for boolean functional synthesis. In *2019 Formal Methods in Computer Aided Design, FMCAD 2019*, pages 161–169, 2019. 42

[ABdR+18] Albert Atserias, Ilario Bonacina, Susanna F. de Rezende, Massimo Lauria, Jakob Nordström, and Alexander A. Razborov. Clique is hard on average for regular resolution. In *50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 866–877, 2018. 37

[ABS15] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015*, pages 56–68, 2015. 44

[ACMS20] Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting knowledge compilation classes and width parameters. *Theory Comput. Syst.*, 64(5):861–914, 2020. 27, 34

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. 17

[AJPU07] Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. *Theory Comput.*, 3(1):81–102, 2007. 37, 110

[AKV04] Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In *Principles and Practice of Constraint Programming, CP 2004*, volume 3258, pages 77–91. Springer, 2004. 36

[AMS18] Antoine Amarilli, Mikaël Monet, and Pierre Senellart. Connecting width and structure in knowledge compilation. In *21st International Conference on Database Theory, ICDT 2018*, volume 98, pages 6:1–6:17, 2018. 34

[AR11] Michael Alekhnovich and Alexander A. Razborov. Satisfiability, branch-width and tseitin tautologies. *Comput. Complex.*, 20(4):649–678, 2011. 37

[Bag06] Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Computer Science Logic, CSL 2006*, pages 167–181, 2006. 43, 44

[BBCP20] Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, and Ján Pich. Frege systems for quantified boolean logic. *J. ACM*, 67(2):9:1–9:36, 2020. 36, 37

[BBH+18] Roderick Bloem, Nicolas Braud-Santoni, Vedad Hadzic, Uwe Egly, Florian Lonsing, and Martina Seidl. Expansion-based QBF solving without recursion. In *Formal Methods in Computer Aided Design, FMCAD 2018*, pages 1–10, 2018. 36

[BBI12] Paul Beame, Christopher Beck, and Russell Impagliazzo. Time-space tradeoffs in resolution: superpolynomial lower bounds for superlinear space. In *44th Symposium on Theory of Computing Conference, STOC 2012*, pages 213–232, 2012. 37

[BCJ19] Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. New resolution-based QBF calculi and their proof complexity. *ACM Trans. Comput. Theory*, 11(4):26:1–26:42, 2019. 36

[BDFG10] Guillaume Bagan, Arnaud Durand, Emmanuel Filiot, and Olivier Gauwin. Efficient enumeration for conjunctive queries over X-underbar structures. In *Computer Science Logic, CSL 2010*, pages 80–94, 2010. 43

[BDG07] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic, CSL 2007*, pages 208–222, 2007. 43

[Ben02] Eli Ben-Sasson. Hard examples for the bounded depth frege proof system. *Comput. Complex.*, 11(3-4):109–136, 2002. 37, 38

[BF21] Beate Bollig and Martin Farenholtz. On the relation between structured d-dnnfs and sdds. *Theory Comput. Syst.*, 65(2):274–295, 2021. 51

[BGS20] Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. 43, 52

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009. 98

[BI13] Christopher Beck and Russell Impagliazzo. Strong ETH holds for regular resolution. In *Symposium on Theory of Computing Conference, STOC'13*, pages 487–494, 2013. 37

[Bie04] Armin Biere. Resolve and expand. In *Seventh International Conference on Theory and Applications of Satisfiability Testing, SAT 2004*, 2004. 36

[BIKS18] Sam Buss, Dmitry Itsykson, Alexander Knop, and Dmitry Sokolov. Reordering rule makes OBDD proof systems stronger. In *33rd Computational Complexity Conference, CCC 2018*, pages 16:1–16:24, 2018. 36

[BJ12] Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods Syst. Des.*, 41(1):45–65, 2012. 37

[BK06] Hans L. Bodlaender and Arie M. C. A. Koster. Safe separators for treewidth. *Discret. Math.*, 306(3):337–350, 2006. 118

[BKM11] Irénée Briquel, Pascal Koiran, and Klaus Meer. On the expressive power of CNF formulas of bounded tree- and clique-width. *Discrete Applied Mathematics*, 159(1):1–14, 2011. 39, 40

[BKS17a] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017*, 2017. 147

[BKS17b] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. In *20th International Conference on Database Theory, ICDT 2017*, pages 8:1–8:18, 2017. 43, 147

[BKS18] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs under updates and in the presence of integrity constraints. In *21st International Conference on Database Theory, ICDT 2018*, pages 8:1–8:19, 2018. 43, 147

[BN21] Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In *Handbook of Satisfiability, 2nd edition*. 2021. 37, 52, 110

[BOW16] Bernhard Bliem, Sebastian Ordyniak, and Stefan Woltran. Clique-width and directed width measures for answer-set programming. In *22nd European Conference on Artificial Intelligence, ECAI 2016*, pages 1105–1113, 2016. 33

[Bra16] Johann Brault-Baron. Hypergraph acyclicity revisited. *ACM Comput. Surv.*, 49(3):54:1–54:26, 2016. 14

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. 21, 23, 24

[BSW02] Beate Bollig, Martin Sauerhoff, and Ingo Wegener. On the Nonapproximability of Boolean Functions by OBDDs and Read-k-Times Branching Programs. *Inf. Comput.*, 178(1):263–278, 2002. 28, 29, 96, 97, 98

[Bul13] Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5):34:1–34:41, 2013. 13

[Cap16] Florent Capelli. *Structural restriction of CNF-formulas: application to model counting and knowledge compilation*. Phd thesis, Université Paris Diderot (Paris 7), Sorbonne Paris Cité, 2016. 27

[ČC20] Ondřej Čepek and Miloš Cromý. Properties of switch-list representations of boolean functions. *J. Artif. Intell. Res.*, 69:501–529, 2020. 22

[CD03] Hei Chan and Adnan Darwiche. Reasoning about Bayesian Network Classifiers. In *Conference in Uncertainty in Artificial Intelligence, UAI 2003*, pages 107–115, 2003. 5, 23, 24

[CD08] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008. 41

[CD13] Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013. 5, 21, 24

[CD17] Arthur Choi and Adnan Darwiche. On relaxing determinism in arithmetic circuits. In *34th International Conference on Machine Learning, ICML 2017*, pages 825–833, 2017. 42

[CDM17] Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 210–223, 2017. 18

[CG10] Hubie Chen and Martin Grohe. Constraint satisfaction with succinctly specified relations. *Journal of Computer and System Sciences*, 76(8):847–860, 2010. 17

[CH96] Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996. 13

[ČH17] Ondřej Čepek and Radek Hušek. Recognition of tractable dnfs representable by a constant number of intervals. *Discret. Optim.*, 23:1–19, 2017. 22

[Che04] Hubie Chen. Quantified constraint satisfaction and bounded treewidth. In *16th Eureopean Conference on Artificial Intelligence, ECAI 2004*, 2004. 33, 34, 35, 39

[Che14a] Hubie Chen. On the complexity of existential positive queries. *ACM Trans. Comput. Log.*, 15(1), 2014. 18

[Che14b] Hubie Chen. The tractability frontier of graph-like first-order query sets. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14*, page 31, 2014. 65, 81

[CK18] Nofar Carmeli and Markus Kröll. Enumeration complexity of conjunctive queries with functional dependencies. In *21st International Conference on Database Theory, ICDT 2018*, pages 11:1–11:17, 2018. 43

[CK19] Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. In *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019*, pages 134–148, 2019. 43

[CKD13] Arthur Choi, Doga Kisa, and Adnan Darwiche. Compiling Probabilistic Graphical Models Using Sentential Decision Diagrams. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU*, pages 121–132, 2013. 24, 41

[CKS01] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 2001. 49

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009. 138

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceddings of STOC'77*, pages 77–90, 1977. 62, 63, 81

[Col07] Thomas Colcombet. A combinatorial theorem for trees. In *34th International ColloquiumAutomata, Languages and Programming, ICALP 2007*, pages 901–912, 2007. 44

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. 34

[CT20] Karine Chubarian and György Turán. Interpretability of Bayesian Network Classifiers: OBDD Approximation and Polynomial Threshold Functions. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM*, 2020. 28, 29, 51, 99, 104

[CVVdB20] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, Technical report, oct 2020. 41

[Dar01a] Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001. 5, 23, 25, 27, 34

[Dar01b] 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):11–34, 2001. 24

[Dar03] Adnan Darwiche. A differential approach to inference in bayesian networks. *J. ACM*, 50(3):280–305, 2003. 41

[Dar04] Adnan Darwiche. New Advances in Compiling CNF into Decomposable Negation Normal Form. In *European Conference on Artificial Intelligence, ECAI*, pages 328–332, 2004. 5, 24

[Dar11] Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, pages 819–826, 2011. 5, 21, 24, 50

[dC20] Alexis de Colnet. A lower bound on DNNF encodings of pseudo-boolean constraints. In *23rd International Conference Theory and Applications of Satisfiability Testing, SAT 2020*, pages 312–321, 2020. 28

[Den16] Aaron W. Dennis. *Algorithms for Learning the Structure of Monotone and Nonmonotone Sum-Product Networks*. PhD thesis, Brigham Young University, 2016. 41

[DG07] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *TOCL*, 8(4), 2007. 43

[DHJ+04] Pavol Duris, Juraj Hromkovic, Stasys Jukna, Martin Sauerhoff, and Georg Schnitger. On multi-partition communication complexity. *Inf. Comput.*, 194(1):49–75, 2004. 26, 29, 30, 40, 99, 100

[DJ04a] V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004. 18

[DJ04b] Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004. 65

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962. 37

[DM02] Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002. 5, 21, 22, 23, 24, 25, 27, 42, 83, 89, 93, 97, 118

[DM07] Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artif. Intell.*, 171(2-3):73–106, 2007. 41

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960. 37, 110

[DPW12] Wolfgang Dvorák, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artif. Intell.*, 186:1–37, 2012. 35

[DR13] Martin E. Dyer and David Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM J. Comput.*, 42(3):1245–1274, 2013. 13

[DSS14] Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2014*, pages 121–131, 2014. 43

[DSW10] Wolfgang Dvorák, Stefan Szeider, and Stefan Woltran. Reasoning in argumentation frameworks of bounded clique-width. In *Computational Models of Argument, COMMA*, pages 219–230, 2010. 33

[Dun07] Paul E. Dunne. Computational properties of argument systems satisfying graph-theoretic constraints. *Artif. Intell.*, 171(10-15):701–729, 2007. 34

[Dur12] David Duris. Some characterizations of γ and β-acyclicity of hypergraphs. *Inf. Process. Lett.*, 112(16):617–620, 2012. 14

[DVVdB20] Meihua Dang, Antonio Vergari, and Guy Van den Broeck. Strudel: Learning structured-decomposable probabilistic circuits. In *10th International Conference on Probabilistic Graphical Models*, pages 137–148, 2020. 41

[Fag83] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983. 14

[FG04] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004. 35

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. 16, 17, 29, 34

[FH18] Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. *Theory Comput. Syst.*, 62(4), 2018. 44

[FHH20] Johannes Klaus Fichte, Markus Hecher, and Florim Hamiti. The model counting competition 2020. *CoRR*, abs/2012.01323, 2020. 4

[FHMW18] Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Exploiting treewidth for projected model counting and its limits. In *21st International Conference Theory and Applications of Satisfiability Testing, SAT 2018*, pages 165–184, 2018. 33, 34

[FHP20] Johannes Klaus Fichte, Markus Hecher, and Andreas Pfandler. Lower bounds for qbfs of bounded treewidth. In *35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '20*, pages 410–424, 2020. 34

[FKP18] Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018*, 2018. 44, 45, 127, 147

[FKRV15] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2), 2015. 44, 45, 123, 127, 140

[FM14] Hélène Fargier and Pierre Marquis. Disjunctive closures for knowledge compilation. *Artif. Intell.*, 216:129–162, 2014. 21

[FMN13] Hélène Fargier, Pierre Marquis, and Alexandre Niveau. Towards a knowledge compilation map for heterogeneous representation languages. In *23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 877–883, 2013. 21

[FMR08] E. Fischer, J.A. Makowsky, and E.V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008. 15, 33, 39

[Fre17] Dominik D. Freydenberger. A logic for document spanners. In *ICDT*, 2017. 44, 45, 124, 125

[Fre19] Dominik D. Freydenberger. A logic for document spanners. *Theory of Computing Systems*, 63(7), 2019. 125

[FRU+18] Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018*, 2018. 5, 44, 45, 46, 123, 125, 126, 129, 147

[FT20] Dominik D. Freydenberger and Sam M. Thompson. Dynamic complexity of document spanners. In *23rd International Conference on Database Theory, ICDT 2020*, pages 11:1–11:21, 2020. 44

[Gal12] François Le Gall. Improved output-sensitive quantum algorithms for boolean matrix multiplication. In *Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 1464–1476, 2012. 126

[Gal14] François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 2014*, pages 296–303, 2014. 126

[GI17] Ludmila Glinskih and Dmitry Itsykson. Satisfiable tseitin formulas are hard for nondeterministic read-once branching programs. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017*, pages 26:1–26:12, 2017. 108

[GIRS19] Nicola Galesi, Dmitry Itsykson, Artur Riazanov, and Anastasia Sofronova. Bounded-depth frege complexity of tseitin formulas for all graphs. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, pages 49:1–49:15, 2019. 38

[GK98] Dima Grigoriev and Marek Karpinski. An exponential lower bound for depth 3 arithmetic circuits. In *Thirtieth Annual ACM Symposium on the Theory of Computing, STOC 1998*, pages 577–582, 1998. 42

[GKM+11] Parikshit Gopalan, Adam R. Klivans, Raghu Meka, Daniel Stefankovic, Santosh S. Vempala, and Eric Vigoda. An FPTAS for #Knapsack and Related Counting Problems. In *IEEE Symposium on Foundations of Computer Science, FOCS 2011*, pages 817–826, 2011. 28, 29, 99

[Goe93] Andreas Goerdt. Regular resolution versus unrestricted resolution. *SIAM J. Comput.*, 22(4):661–683, 1993. 37

[GP04] Georg Gottlob and Reinhard Pichler. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. *SIAM J. Comput.*, 33(2):351–378, 2004. 14

[GPW10] Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010. 34, 35, 39

[Gra96] Étienne Grandjean. Sorting, linear time and the satisfiability problem. *Annals of Mathematics and Artificial Intelligence*, 16(1), 1996. 132

[Gro07] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007. 18, 65

[GSS01] Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *ACM Symposium on Theory of Computing, STOC 2001*, 2001. 18

[GTT20] Nicola Galesi, Navid Talebanfard, and Jacobo Torán. Cops-robber games and the resolution of tseitin formulas. *ACM Trans. Comput. Theory*, 12(2):9:1–9:22, 2020. 37, 38, 120

[HCD06] Jinbo Huang, Mark Chavira, and Adnan Darwiche. Solving MAP exactly by searching on compiled arithmetic circuits. In *Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, pages 1143–1148, 2006. 41

[HD07] Jinbo Huang and Adnan Darwiche. The language of search. *J. Artif. Intell. Res.*, 29:191–219, 2007. 5, 25

[HPSS18] Holger H. Hoos, Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Portfolio-based algorithm selection for circuit qbfs. In *24th International ConferencePrinciples and Practice of Constraint Programming, CP 2018*, pages 195–209, 2018. 36

[Hro97] Juraj Hromkovic. *Communication Complexity and Parallel Computing.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1997. 40

[HW17] Daniel J. Harvey and David R. Wood. Parameters tied to treewidth. *J. Graph Theory*, 84(4):364–385, 2017. 107

[IO13] Dmitry Itsykson and Vsevolod Oparin. Graph expansion, tseitin formulas and resolution proofs for CSP. In *8th International Computer Science Symposium in Russia, CSR 2013*, pages 162–173, 2013. 37

[IRSS19] Dmitry Itsykson, Artur Riazanov, Danil Sagunov, and Petr Smirnov. Almost tight lower bounds on regular resolution refutations of tseitin formulas for all constant-degree graphs. *Electron. Colloquium Comput. Complex.*, 26:178, 2019. 37, 38, 52, 109, 110, 111, 112

[IW10] Russell Impagliazzo and Ryan Williams. Communication complexity with synchronized clocks. In *25th Annual IEEE Conference on Computational Complexity, CCC 2010*, pages 259–269, 2010. 37

[JBKW08] Jean Christoph Jung, Pedro Barahona, George Katsirelos, and Toby Walsh. Two encodings of DNNF theories. In *ECAI workshop on Inference methods based on Graphical Structures of Knowledge*, pages 1891–1897, 2008. 5

[JKMC16] Mikolás Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, 234:1–25, 2016. 36

[JM15] Mikolás Janota and João Marques-Silva. Solving QBF by clause selection. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 325–331, 2015. 36

[JPRW08] Michael Jakl, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Fast counting with bounded treewidth. In *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2008*, pages 436–450, 2008. 35

[JPW09] Michael Jakl, Reinhard Pichler, and Stefan Woltran. Answer-set programming with bounded treewidth. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, 2009. 39

[JS82] Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *J. ACM*, 29(3):874–897, 1982. 41

[JS02] Stasys Jukna and Georg Schnitger. Triangle-freeness is hard to detect. *Combinatorics, Probability & Computing*, 11(6):549–569, 2002. 26, 90

[JS13] Abhay Kumar Jha and Dan Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.*, 52(3):403–440, 2013. 26, 51

[Juk12] Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. 25, 40, 92

[KCL+19] Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. On tractable computation of expected predictions. In *Annual Conference on Neural Information Processing Systems, NeurIPS 2019*, pages 11167–11178, 2019. 41

[KdBCD14] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Fourteenth International Conference Principles of Knowledge Representation and Reasoning, KR 2014*, 2014. 41, 42

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997. 26, 37, 40, 93, 97

[Kri06] Matthias P. Krieger. *Some Restricted Circuit Classes that Can Be as Inefficient as DNFs*. Diploma thesis, Johann Wolfgang Goethe-Universität, Germany, Frankfurt am Main, 2006. 26

[KS13a] Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2013*, pages 297–308, 2013. 43

[KS13b] Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *TOCL*, 14(4), 2013. 43, 44

[KS19] Petr Kucera and Petr Savický. Propagation complete encodings of smooth DNNF theories. *CoRR*, abs/1909.06673, 2019. 5

[KST97] S. Khanna, M. Sudan, and L. Trevisan. Constraint satisfaction: The approximability of minimization problems. In *12th IEEE Conference on Computational Complexity (CCC'97)*, pages 282–296, 1997. 49

[KSW99] Matthias Krause, Petr Savický, and Ingo Wegener. Approximations by OBDDs and the Variable Ordering Problem. In *International Colloquium Automata, Languages and Programming, ICALP 1999*, pages 493–502, 1999. 28, 95, 96

[Lam10] Michael Lampis. Algorithmic Meta-theorems for Restrictions of Treewidth. In *18th Annual European Symposium on Algorithms, ESA 2010*, pages 549–560, 2010. 30

[LB10] Florian Lonsing and Armin Biere. Depqbf: A dependency-aware QBF solver. *J. Satisf. Boolean Model. Comput.*, 7(2-3):71–76, 2010. 36

[LD08] Daniel Lowd and Pedro M. Domingos. Learning arithmetic circuits. In *24th Conference in Uncertainty in Artificial Intelligence, UAI 2008*, pages 383–392, 2008. 41

[LE18] Florian Lonsing and Uwe Egly. Evaluating QBF solvers: Quantifier alternations matter. In *24th International Conference Principles and Practice of Constraint Programming, CP 2018*, pages 276–294, 2018. 36

[LM14] Katja Losemann and Wim Martens. MSO queries on trees: Enumerating answers under updates. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14*, 2014. 147

[LM17a] Jean-Marie Lagniez and Pierre Marquis. An Improved Decision-DNNF Compiler. In *International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 667–673, 2017. 5, 24

[LM17b] Michael Lampis and Valia Mitsou. Treewidth with a quantifier alternation revisited. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017*, pages 26:1–26:12, 2017. 35

[LMS11] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. 35

[LNNW95] László Lovász, Moni Naor, Ilan Newman, and Avi Wigderson. Search problems in the decision tree model. *SIAM J. Discret. Math.*, 8(1):119–132, 1995. 38, 110

[LRRS14] Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Practical algorithms for MSO model-checking on tree-decomposable graphs. *Computer Science Review*, 13-14:39–74, 2014. 35

[LS81] Richard J. Lipton and Robert Sedgewick. Lower bounds for VLSI. In *13th Annual ACM Symposium on Theory of Computing, STOC 1981*, pages 300–307, 1981. 91

[LT80] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. 40

[Mar10a] Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. 70

[Mar10b] Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 735–744, 2010. 81

[Mar15] Pierre Marquis. Compile! In *Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015*, pages 4112–4118, 2015. 4

[Men13] Stefan Mengel. *Conjunctive queries, arithmetic circuits and counting complexity*. PhD thesis, University of Paderborn, 2013. 17

[MM14] James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks. *CoRR*, abs/1411.7717, 2014. 42

[MM16] Dániel Marx and Valia Mitsou. Double-exponential and triple-exponential bounds for choosability problems parameterized by treewidth. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, pages 28:1–28:15, 2016. 35

[MMBH12] Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-DNNF Compilation with sharpSAT. In *Canadian Conference on Artificial Intelligence*, pages 356–361, 2012. 5, 24

[Mon20] Mikaël Monet. Solving a special case of the intensional vs extensional conjecture in probabilistic databases. In *39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020*, pages 149–163, 2020. 5, 51

[Mor17] Andrea Morciano. Engineering a runtime system for AQL. Master's thesis, Politecnico di Milano, 2017. 142, 145, 146

[MRV18] Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document spanners for extracting incomplete information: Expressiveness and complexity. In *SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018*, 2018. 44, 45, 123, 124, 125, 127

[MS16] Arnaud Mary and Yann Strozecki. Efficient enumeration of solutions produced by closure operations. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, pages 52:1–52:13, 2016. 46

[Nie18] Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras. In *ACM/IEEE Symposium on Logic in Computer Science (LICS), LICS 2018*, 2018. 147

[Nor15] Jakob Nordström. On the interplay between proof complexity and SAT solving. *ACM SIGLOG News*, 2(3):19–44, 2015. 37

[NS18] Matthias Niewerth and Luc Segoufin. Enumeration of MSO queries on strings with constant delay and logarithmic updates. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018*, 2018. 147

[OD15] Umut Oztok and Adnan Darwiche. A Top-Down Compiler for Sentential Decision Diagrams. In *International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 3141–3148, 2015. 5, 24

[OD17] Umut Oztok and Adnan Darwiche. On compiling dnnfs without determinism. *CoRR*, abs/1709.07092, 2017. 51

[OPS13] Sebastian Ordyniak, Daniël Paulusma, and Stefan Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theor. Comput. Sci.*, 481:85–99, 2013. 14, 15

[OZ15] Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst. (TODS)*, 40(1):2:1–2:44, 2015. 23, 43, 44

[PD08] Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 517–522, 2008. 21, 24, 27, 33, 43

[PD10] Thammanit Pipatsrisawat and Adnan Darwiche. A Lower Bound on the Size of Decomposable Negation Normal Form. In *AAAI Conference on Artificial Intelligence, AAAI 2010*, pages 345–350, 2010. 27, 34, 40, 83, 91, 92, 93

[PD11] Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture. In *IEEE International Conference on Computer Vision Workshops, ICCV 2011*, pages 689–690, 2011. 41

[Pet21] Liat Peterfreund. Grammars for document spanners. In *24th International Conference on Database Theory, ICDT 2021*, pages 7:1–7:18, 2021. 44

[PFKK19] Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019*, pages 320–334, 2019. 44

[PSS13] Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model Counting for CNF Formulas of Bounded Modular Treewidth. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013*, pages 55–66, 2013. 15, 29

[PSS16] Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016. 33, 39

[PSS19] Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. *J. Artif. Intell. Res.*, 65:180–208, 2019. 36

[PT09] Luca Pulina and Armando Tacchella. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints An Int. J.*, 14(1):80–116, 2009. 36

[PTPD15] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro M. Domingos. On theoretical properties of sum-product networks. In *Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015*, 2015. 41

[PV04] Guoqiang Pan and Moshe Y. Vardi. Symbolic decision procedures for QBF. In *10th International Conference Principles and Practice of Constraint Programming, CP 2004*, pages 453–467, 2004. 36

[PV06] Guoqiang Pan and Moshe Y. Vardi. Fixed-parameter hierarchies inside PSPACE. In *21th IEEE Symposium on Logic in Computer Science, LICS 2006*, pages 27–36, 2006. 34

[PY99] C. Papadimitriou and M. Yannakakis. On the Complexity of Database Queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999. 17

[Raz09] Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2):8:1–8:17, 2009. 41, 42

[Raz10] Ran Raz. Elusive functions and lower bounds for arithmetic circuits. *Theory Comput.*, 6(1):135–177, 2010. 42

[Raz14] Igor Razgon. No small nondeterministic read-once branching programs for cnfs of bounded treewidth. In Marek Cygan and Pinar Heggernes, editors, *9th International SymposiumParameterized and Exact Computation, IPEC 2014*, pages 319–331, 2014. 27

[Raz16] Igor Razgon. On the read-once property of branching programs and cnfs of bounded treewidth. *Algorithmica*, 75(2):277–294, 2016. 27, 114

[Res18] IBM Research. SystemT, 2018. 44

[RKG14] Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *European Conference Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2014*, pages 630–645, 2014. 41

[RL16] Amirmohammad Rooshenas and Daniel Lowd. Discriminative structure learning of arithmetic circuits. In *Thirtieth AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 4258–4259, 2016. 41

[Rom18] Miguel Romero. The tractability frontier of well-designed SPARQL queries. In *37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 295–306, 2018. 50

[Rot96] Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996. 13

[RT75] Ronald C. Read and Robert E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3), 1975. 46

[RT15] Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In *Formal Methods in Computer-Aided Design, FMCAD 2015*, pages 136–143, 2015. 36

[RY08] Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Comput. Complex.*, 17(4):515–535, 2008. 23

[Sau03] Martin Sauerhoff. Approximation of boolean functions by combinatorial rectangles. *Theor. Comput. Sci.*, 1-3(301):45–78, 2003. 27, 50, 89

[SCD19] Andy Shih, Arthur Choi, and Adnan Darwiche. Compiling Bayesian Network Classifiers into Decision Graphs. In *AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 7966–7974, 2019. 24

[SdBBA19] Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, pages 11412–11422, 2019. 24

[Seg14] Luc Segoufin. A glimpse on constant delay enumeration (invited talk). In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, pages 13–27, 2014. 43

[SGZ05] Baruch Schieber, Daniel Geist, and Ayal Zaks. Computing the minimum DNF representation of boolean functions defined by intervals. *Discret. Appl. Math.*, 149(1-3):154–173, 2005. 22

[SK96] Bart Selman and Henry A. Kautz. Knowledge Compilation and Theory Approximation. *J. ACM*, 43(2):193–224, 1996. 29

[SKAP06] Jonathan Schler, Moshe Koppel, Shlomo Argamon, and James W Pennebaker. Effects of age and gender on blogging. In *AAAI spring symposium: Computational approaches to analyzing weblogs*, pages 199–205, 2006. 142, 145

[Som09] Fabio Somenzi. CUDD: CU decision diagram package-release 2.4. 0. *University of Colorado at Boulder*, 2009. 21

[SS10a] Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010. 14, 15, 33, 39

[SS10b] Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010. 40

[SS13] Friedrich Slivovsky and Stefan Szeider. Model counting for formulas of bounded clique-width. In *24th International Symposium Algorithms and Computation, ISAAC 2013*, pages 677–687, 2013. 14, 15, 33, 39

[SS21] Markus L. Schmid and Nicole Schweikardt. A purely regular approach to non-regular core spanners. In *24th International Conference on Database Theory, ICDT 2021*, pages 4:1–4:19, 2021. 44

[SSV18] Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In *37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018*, pages 151–163, 2018. 43

[STV15] Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving #sat and MAXSAT by dynamic programming. *J. Artif. Intell. Res.*, 54:59–82, 2015. 15, 16, 33, 39

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5(3-4):207–388, 2010. 23, 41, 42

[Sze04] Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In *6th International Conference on Theory and Applications of Satisfiability, SAT 2004*, pages 188–202, 2004. 33

[Ten16] Leander Tentrup. Non-prenex QBF solving using abstraction. In *19th International Conference on Theory and Applications of Satisfiability Testing, SAT 2016*, pages 393–401, 2016. 36

[TIAS77] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and I Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6, 09 1977. 46

[TNY97] Yasuhiko Takenaga, Mitsushi Nouzoe, and Shuzo Yajima. Size and Variable Ordering of OBDDs Representing Treshold Functions. In *International Conference on Computing and Combinatorics, COCOON 1997*, pages 91–100, 1997. 28, 99

[Tod91] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. 13

[Tse68] G.S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, Part 2:115–125, 1968. 37

[Urq87] Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987. 37, 108

[Urq11] Alasdair Urquhart. A near-optimal separation of regular and general resolution. *SIAM J. Comput.*, 40(1):107–121, 2011. 37, 110

[Val79a] Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. 4

[Val79b] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2), 1979. 46

[Val80] Leslie G. Valiant. Negation can be exponentially powerful. *Theor. Comput. Sci.*, 12:303–314, 1980. 41

[Vat12] Martin Vatshelle. *New Width Parameters of Graphs*. Phd-thesis, University of Bergen, 2012. 16

[VCL+21] Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations: From simple transformations to complex information-theoretic queries. *CoRR*, abs/2102.06137, 2021. 41

[VEJN20] Marc Vinyals, Jan Elffers, Jan Johannsen, and Jakob Nordström. Simplified and improved separations between regular and general resolution by lifting. In *23rd International Conference on Theory and Applications of Satisfiability Testing, SAT 2020*, pages 182–200, 2020. 37

[Wal20] Romain Wallon. *Pseudo-Boolean Reasoning and Compilation*. Phd thesis, Université d'Artois, Lens, France, 2020. 49

[Was16] Kunihiro Wasa. Enumeration of enumeration algorithms. *CoRR*, abs/1605.05102, 2016. 42

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000. 25, 40, 43

[Wil18] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3447–3487. World Scientific, 2018. 52

[ZM02] Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002*, pages 442–449, 2002. 36