Parametrised Enumeration

Arne Meier

Februar 2020

Gottfried Wilhelm Leibniz Universität Hannover

Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik
Fachgebiet Theoretische Informatik

Habilitationsschrift

Parametrised Enumeration

Arne Meier
geboren am 6. Mai 1982 in Hannover

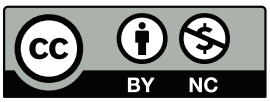
Februar 2020

Gutachter Till Tantau
Institut für Theoretische Informatik
Universität zu Lübeck

Gutachter Stefan Woltran
Institut für Logic and Computation 192-02
Technische Universität Wien

Arne Meier
Parametrised Enumeration
Habilitationsschrift, Datum der Annahme: 20.11.2019
Gutachter: Till Tantau, Stefan Woltran

Gottfried Wilhelm Leibniz Universität Hannover
Fachgebiet Theoretische Informatik
Institut für Theoretische Informatik
Fakultät für Elektrotechnik und Informatik
Appelstrasse 4
30167 Hannover



Für Julia,
Jonas Heinrich und Leonie
Anna. Ihr seid mein größtes
Glück auf Erden. Danke für eure
Geduld, euer Verständnis und
eure Unterstützung. Euer
Rückhalt bedeutet
mir sehr viel.
♡

> *If I had an hour to solve a problem, I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions.*
>
> **— Albert Einstein**

Abstract

In this thesis, we develop a framework of parametrised enumeration complexity. At first, we provide the reader with preliminary notions such as machine models and complexity classes besides proving them to be well-chosen. Then, we study the interplay and the landscape of these classes and present connections to classical enumeration classes. Afterwards, we translate the fundamental methods of kernelisation and self-reducibility into equivalent techniques in the setting of parametrised enumeration. Subsequently, we illustrate the introduced classes by investigating the parametrised enumeration complexity of Max-Ones-SAT and strong backdoor sets as well as sharpen the first result by presenting a dichotomy theorem for Max-Ones-SAT. After this, we extend the definitions of parametrised enumeration algorithms by allowing orders on the solution space. In this context, we study the relations "order by size" and "lexicographic order" for graph modification problems and observe a trade-off between enumeration delay and space requirements of enumeration algorithms. These results then yield an enumeration technique for generalised modification problems that is illustrated by applying this method to the problems closest string, weak and strong backdoor sets, and weighted satisfiability. Eventually, we consider the enumeration of satisfying teams of formulas of poor man's propositional dependence logic. There, we present an enumeration algorithm with FPT delay and exponential space which is one of the first enumeration complexity result of a problem in a team logic. Finally, we show how this algorithm can be modified such that only polynomial space is required, however, by increasing the delay to incremental FPT time.

Keywords: Parametrised Complexity, Enumeration Complexity, Parametrised Enumeration

Zusammenfassung

In diesem Werk begründen wir die Theorie der parametrisierten Enumeration, präsentieren die grundlegenden Definitionen und prüfen ihre Sinnhaftigkeit. Im nächsten Schritt, untersuchen wir das Zusammenspiel der eingeführten Komplexitätsklassen und zeigen Verbindungen zur klassischen Enumerationskomplexität auf. Anschließend übertragen wir die zwei fundamentalen Techniken der Kernelisierung und Selbstreduzierbarkeit in Entsprechungen in dem Gebiet der parametrisierten Enumeration. Schließlich untersuchen wir das Problem Max-Ones-SAT und das Problem der Aufzählung starker Backdoor-Mengen als typische Probleme in diesen Klassen. Die vorherigen Resultate zu Max-Ones-SAT werden anschließend in einem Dichotomie-Satz vervollständigt. Im nächsten Abschnitt erweitern wir die neuen Definitionen auf Ordnungen (auf dem Lösungsraum) und erforschen insbesondere die zwei Relationen „Größenordnung“ und „lexikographische Reihenfolge“ im Kontext von Graphen-Modifikationsproblemen. Hierbei scheint es, als müsste man zwischen Delay und Speicheranforderungen von Aufzählungsalgorithmen abwägen, wobei dies jedoch nicht abschließend gelöst werden kann. Aus den vorherigen Überlegungen wird schließlich ein generisches Enumerationsverfahren für allgemeine Modifikationsprobleme entwickelt und anhand der Probleme Closest String, schwacher und starker Backdoor-Mengen sowie gewichteter Erfüllbarkeit veranschaulicht. Im letzten Abschnitt betrachten wir die parametrisierte Enumerationskomplexität von Erfüllbarkeitsproblemen im Bereich der Poor Man’s Propositional Dependence Logic und stellen einen Aufzählungsalgorithmus mit FPT Delay vor, der mit exponentiellem Platz arbeitet. Dies ist einer der ersten Aufzählungsalgorithmen im Bereich der Teamlogiken. Abschließend zeigen wir, wie dieser Algorithmus so modifiziert werden kann, dass nur polynomieller Speicherplatz benötigt wird, bezahlen jedoch diese Einsparung mit einem Anstieg des Delays auf inkrementelle FPT Zeit (IncFPT).

Schlagwörter: Parametrisierte Algorithmen, Aufzählungskomplexität, Parametrisierte Aufzählungsalgorithmen

Acknowledgement

At first, I wish to deeply thank my family for always being sympathetic and supportive. Secondly, I want to express my gratitude to all my coauthors of publications which have been incorporated in this thesis. In particular, I show appreciation to Nadia Creignou, Raïda Ktari, Julian-Steffen Müller, Fréderic Olive, Johannes Schmidt (thanks, Johannes, for the discussion on Theorem 3.2), and Heribert Vollmer who have been working with me at the initial steps into this exciting new field. Furthermore, I thank Maurice Chandoo and Anselm Haak for their valuable comments on various parts of this thesis. Particularly, I would like to thank Martin Lück for his feedback on Section 3.2, and Johannes Fichte for proof reading Chapter 1. I want to give thanks to Christian Reinbold for discussions on Chapter 6. Eventually, I appreciate Anselm's feedback to my teaching demonstration and Fabian's on my scientific talk. Finally, I wish to thank Stefan Woltran, and Till Tantau for reviewing this thesis.

Danksagung

Zuerst möchte ich meine tiefste Dankbarkeit bei meiner Familie ausdrücken. Euer Verständnis und eure Unterstützung in allen Lebenslagen sind das Fundament meiner Forschung. An zweiter Stelle bedanke ich mich bei allen meinen Mitautoren von Veröffentlichungen, die in diese Arbeit eingeflossen sind. Besonderer Dank gehört hierbei Nadia Creignou, Raïda Ktari, Julian-Steffen Müller, Fréderic Olive, Johannes Schmidt (besonderer Dank geht an Dich für die Diskussion zu Theorem 3.2) und Heribert Vollmer, die bei den wichtigen ersten Publikationen zur parametrisierten Enumeration mitgewirkt haben. Schließlich richtet sich meine Dankbarkeit an Maurice Chandoo und Anselm Haak für ihre wertvollen Rückmeldungen zu unterschiedlichen Abschnitten dieser Arbeit. Insbesondere möchte ich Martin Lück für seine Hinweise zum Abschnitt 3.2 sowie Johannes Fichte für das Korrekturlesen von Kapitel 1 danken. Christian Reinbold bin ich für die Diskussionen zu Kapitel 6 zu Dank verpflichtet. Schließlich danke ich Anselm für seine Rückmeldung zu meiner Lehrprobe und Fabian für seine Tipps zu meinem wissenschaftlichen Vortrag. Abschließend möchte ich mich noch bei Stefan Woltran und Till Tantau für das begutachten dieser Arbeit bedanken.

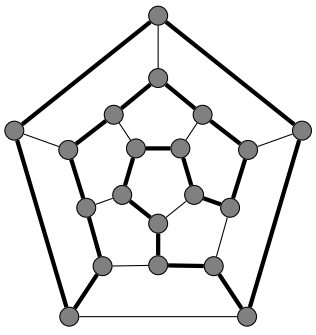
Contents

1 Introduction

> *[...]; les charmes enchanteurs de cette sublime science ne se decelent dans toute leur beaute qu'a ceux qui ont le courage de l'approfondir.*
>
> *[...]; the enchanting charms of this sublime science reveal themselves in all their beauty only to those who have the courage to go deeply into it.*
>
> **— Carl Friedrich Gauß, 30. April 1807.**
>
> [CF12]

In the last decade, implicitly using modern technologies silently became a common procedure in copious areas of living. Before leaving home, we check the rain forecast on our smartphone. In the morning, we buy groceries or various other things over the internet, and they are delivered even the same day. Driving a car to a new place without using a navigation system and relying solely on a printed map instead seems absurd these days. However, often, we do not realise how complex and striking the development of the underlying algorithmic tasks has been.

The process of finding an (algorithmic) solution lies in the heart of theoretical computer science. Despite the variety of different techniques that are utilised, we want to exemplify this richness slightly more detailed in the following. Consider the problem of finding a shortest possible route visiting cities from a given list and each of them exactly once. This task is well-known under the name the *travelling salesperson problem* (TSP), whose origin lies in the work of William Rowan Hamilton and his *icosian game* [Ham53]. In this game, one is given a dodecahedron, which is a polyhedron with twelve flat faces (or also a twelve sided dice):



In this specific graph, one searches for a round trip of the mentioned type that also is a cycle. Historically, such routes are called *Hamiltonian* today, and the thick edges depict a possible one.

But now, let us turn back to the problem of the travelling salesperson. The mere question of whether a journey shorter than a given cost exists, gained a wealth

of attention by researchers in many different subareas of computer science and mathematics. The foundational work on TSP developed many remarkable algorithmic techniques (for instance, integer programming [Bea65], cutting-plain method [Mar+02]) and also improved the view of the landscape of computational complexity theory drastically. In this area, one is interested in measuring the required runtime and memory that is needed to solve the problem dependent on a formal computation model. In 1972, Richard Karp proved in his seminal publication "Reducibility Among Combinatorial Problems" [Kar72] (among 21 other problems) the NP-completeness of a very closely related problem (finding Hamiltonian paths in graphs) and thereby settled the intractability of TSP. Intuitively speaking, NP-completeness of a problem interdicts the existence of an algorithm for that particular problem (and accordingly for TSP) running in polynomial time (unless the complexity classes NP and P coincide which would be a dramatic and quite unlikely event). Here, the complexity class P encompasses all problems solvable in polynomial time and NP those which can be solved in *nondeterministic* polynomial time. Comparing both classes, one can show that for problems in the class P their solutions can be efficiently detected (that is, in polynomial time with respect to the input length), whereas, for NP, solutions can only be efficiently *verified*. It is worth asking in this context how P and NP exactly relate to each other, that means, either $\mathsf{P} = \mathsf{NP}$ is true or P is a strict subclass of NP. Unfortunately, since the 1970s, this remains to be an open question of research. Today, we still do not know whether efficiently verifying solutions also always allows for efficiently finding them. This question is well-known as the prominent P-versus-NP problem. It belongs to the "Millennium Prize Problems" [Cla] which is a list of problems advertised by the Clay Mathematics Institute. This list contains the seven most important, open problems in mathematics. Solving For solving problems on this list, the institute has awarded one million US-dollar in prize money. So far only the Poincaré conjecture has been proved [Per03]. Solving the P-versus-NP problem positively, that is, showing $\mathsf{P} = \mathsf{NP}$, would yield severe consequences to not only the computation of efficient routing solutions but would also imply that vast parts of the currently used cryptographic protocols on the internet are insecure. One reason for this lies in the meaning of a *completeness* result. Essentially, every NP-complete problem can be seen as a representative of this complexity class NP: Every other problem in this class NP can be efficiently translated (in some way) into this particular problem. As a result, if we can solve this specific problem in polynomial time then we can do so as well for every other problem in NP. This fact led to a long list of attempts to solve this problem (positively or negatively) as reported by Gerhard Woeginger on his website `https://www.win.tue.nl/~gwoegi/P-versus-NP.htm`. We want to emphasise that this website only contains *attempts* and none of them has solved this major question yet. Nevertheless, we all know that today, carrier companies still succeed in delivering their packages to their customers. As a consequence, people who are not familiar with complexity theory might question the practical relevance of such theoretical results. At first, one has to know that the quality of the

transport routes which currently are produced is suboptimal—the carrier computes only a "close to optimal" route. Let us define "close to optimal" more precisely in the following. Christofides [Chr76] attacked this problem in the framework of *approximation algorithms*. Here, one constructs deterministic procedures which aim for solutions that are very close to an optimal one. By a clever approach, Christofides constructed an algorithm which guarantees to find a route that is at most 1.5 times longer than an optimal one (since 2014, there exists an improvement to 1.4 [SV14]).

As an interesting fact, humans are very good in spotting (close to) optimal solutions and it is very difficult to translate this intuition into an algorithmic scheme. This fact sometimes results in humans as a last step in the process of spotting solutions efficiently [Enc17].

As we have seen, the NP-completeness of a problem theoretically and practically interdicts the existence of algorithms solving it and running in polynomial time unless $P = NP$. Nonetheless, science has found many other ways to attack such problems today; constructing approximation algorithms is only one of them. This can also be seen as the crucial point of Woeginger's website. Although all of the presented attempts failed in proving either result, the research on the P-versus-NP problem still is indispensable as it spawns new techniques and insights into the heart of this critical problem.

1.1 Parametrised Complexity

One further intriguing approach to cope with such problems is to perform a *fine-grained* complexity analysis by considering their so-called parametrised complexity. Downey and Fellows—who can be seen as initiators of this field [DF13; DF99]—argue that "*parameterized complexity [...] is more attuned to analysing computational questions which arise in practice than traditional worst-case analysis.*" By their statement, they underline how the coarseness of (traditional) worst-case complexity and NP-completeness results might obfuscate feasible solutions for restrictions of the considered problem. In parametrised complexity, the *structure* of the problem is of utmost interest:

> The complexity of a problem usually *not only* relies on the size of the input!

Instead, many problems exhibit so-called *parameters* (or *parametrisations*) which are the true source of complexity. Formally, such parameters are functions mapping an input often to a natural number and are denoted with the Greek letter *Kappa* κ. Spotting such parameters is a key task in reaching deeper views on the problem of interest. In the succeeding process of analysing the parametrised complexity of the given problem, we want to elaborate on two relevant types of studied algorithmic runtimes. On one hand, runtimes of the form $f(\kappa(x)) \cdot p(|x|)$ are much desired, where f is an arbitrary computable function, κ is the parametrisation, and x is the input. By contrast, one usually tries to avoid runtimes of the kind $n^{f(\kappa(x))}$. An obvious objection here is that f, as an arbitrary computable function, could grow so fast

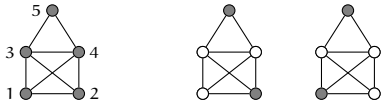
that it renders both of the aforementioned runtimes infeasible. While this is true in theory, these parameters are often of a much smaller value in practice. Typically, one aims for parameters that are growing slowly or even constant [AFN04]. Then one could argue that, under worst-case complexity notion, algorithms with both runtimes solve the problem efficiently (in polynomial time). Nevertheless, a runtime of n^6 or higher is not really of a practical use. By contrast, a runtime of $6 \cdot n$ is a quite fast linear time algorithm.

After we saw that distinguishing these two types of runtimes is sensible, let us turn back to the central observation from above about the structure of the investigated problems. Clearly, this structure is a manifold aspect highly depending on the problem of interest. Often, it is very tedious to detect a parameter which is severely influencing the runtime and, on the other hand, also relevant for practice. For instance, if one considers the satisfiability problem of propositional formulas (SAT), the number of variables is a very weak parameter: Typically, this parameter is neither bounded nor slowly growing, albeit the problem SAT can be solved in time $2^k \cdot |\varphi|$ where k is the number of variables of a given formula φ. As a consequence, Samer and Szeider [SS09] searched for a more fruitful parameter. It turned out that such a parameter is *treewidth* [BK08] encoding the *tree-likeliness* of a graph (the smaller the parametric value the closer the graph is to being a tree). Samer and Szeider proved that SAT parametrised by its *treewidth* (for specific graph representations of input formulas) is *fixed-parameter tractable* (FPT), that is, of the aforementioned much desired runtime $f(\kappa(x)) \cdot p(|x|)$.

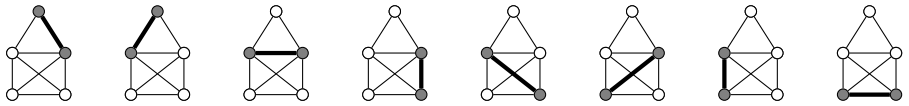
A quantified Boolean formula (QBF) is an extension of propositional formulas by combining existential ($\exists$) and universal ($\forall$) quantifiers with variables. The semantics require that subsequent formulas have to be fulfilled for one ($\exists$) or both ($\forall$) truth values. Such formulas allow to express several problems relevant for practice (such as verification tasks and consistency questions of computer programs) that are presumably of even higher complexity than SAT. Now, similarly, recent research [Fic+17; CW16a; CW16b] shows that graph representations of real-world instances of QBFs seem to exhibit hidden structures not appearing in randomly generated instances. Again, the parameter *treewidth* describes these patterns appropriately. Earlier, Gottlob et al. [GSS02] showed that some specific restrictions of QBFs parametrised by treewidth are in FPT, as well. However, for general QBF formulas, one has to additionally incorporate the quantifier alternations into the parameter to reach FPT as shown by Pan and Vardi [PV06].

Similarly to the work on TSP, the search for parametrised algorithms has yielded numerous new astounding techniques such as kernelisation [DF99], search trees and branching algorithms [HNW08; ST08], iterative compression [RSV04], or colour coding [AYZ95; Cyg+15]. Let us exemplify one of these techniques to provide a better understanding of what parametrised complexity really is about. The method we want to elaborate on is kernelisation, which is best explained via the *vertex-cover problem* (VERTEX-COVER). A *vertex-cover* of a given undirected graph is a (sub-)set

of its vertices such that every edge of the graph is incident to a vertex from the cover. As an example, let us consider the following undirected graph with its vertex covers of size 3, both emphasised on the right-hand side by white filling:

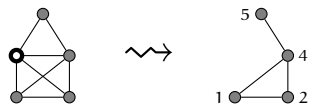


The remaining combinations, where the cover is again filled in white, are no vertex-covers as the thick lines indicate an uncovered edge.



The problem VERTEX-COVER now asks, given an undirected graph G and a natural number $k \in \mathbb{N}$, if G has a vertex-cover of size $\leq k$. VERTEX-COVER is among the prominent 21 problems of Richard Karp [Kar72] and was classified as NP-complete. What we have seen so far lets us deduce that VERTEX-COVER is an intractable problem. In the following, we will analyse the problem in a more detailed way and will compute a kernelisation of the considered instance. Suppose we are given the above graph, the integer $k = 3$, and we need to decide whether this instance belongs to VERTEX-COVER or not.

How can we find an answer to the question in a systematic way? How does the inherent structure of the graph helps us? Firstly, the vertices with "high" degree are important: If there is a vertex present with a degree $d > 3$ it must be part of the vertex cover because otherwise, we can only cover at most $d - 3$ of its edges. In our case, the only two vertices of this kind are number 3 and 4. So, let us choose for vertex 3. *What is the next step?* Now, that we have decided upon vertex 3, we cover all its edges to the vertices 1, 2, 4, and 5. Accordingly, the degrees of these adjacent vertices decreased by one or, equivalently, we remove 3 from the graph and decrease k by one to 2.



Now, that vertex 4 has a degree of 3, comparing it to the updated value of $k = 2$, this still is a high degree leading to selecting vertex 4 next.

In this last step, k reaches the value of 1 and every remaining vertex has a degree of either 1 or 0. Accordingly, there exists no vertex with a high degree. Moreover, we are left with only a single edge. For this graph computing a vertex cover of size 1

is very easy. The general algorithm would stop here by returning the reached graph and $k = 1$ (in the general case, the algorithm would not return graphs with more than k^2 many edges, and instead output a trivial no-instance).

What have we done now? Essentially, after having iteratively selected mandatory vertices, we reached a threshold size of the graph which can be brute-forced in the size of the parameter. More formally, we computed a smaller subgraph G' and a new value k' in time linear in k plus the size of the original graph G. For this subinstance (G', k') we have that $k' \leq k$ and the size of G' is bounded quadratically in k'.

Let us shortly reflect on the presented algorithmic paradigm. In our case, we have computed the prominent Buss kernelisation[1], which, in fact, is even a polynomial kernelisation (that is, the kernel has polynomial size; here quadratic). There even exists also a linear kernelisation by Nemhauser and Trotter [NT75] which we do not investigate further.

Kernelisations provide a compression of the original problem instance to its "kernel" which often is a subinstance like in our case a subgraph. Usually, the size of this subinstance is eminently reduced such that brute-forcing it often is a valid option. In our case, we have seen that VERTEX-COVER parametrised by the value of k is in FPT via the sketched kernelisation. Accordingly, the technique of kernelisation can be seen as an efficient notion of preprocessing. We will later in Section 4.1 return to this technique and show how it can be employed to characterise a new complexity class.

The generalised version of VERTEX-COVER, known as the *hitting-set problem*, has practical applications for dynamically detecting potential data races in multithreaded programs [OC03]. Besides, it has a long tradition in parametrised complexity theory [DF13; FG06; Nie06].

Now, that we have provided an insight into the area of parametrised complexity, we leave the category of decision problems and turn on towards the other central area of this thesis: Enumeration complexity.

1.2 Enumeration

Completely different to deciding the existence of a satisfying assignment of a propositional formula is the algorithmic task of outputting *all* satisfying ones. This type of procedure is well-known under the term *enumeration* (of objects) and is a significant problem in the areas of combinatorics, computational geometry, and operations research [AF96]. Moreover, recent results unveil major importance in web search, data mining, bioinformatics, and computational linguistics [COS11]. Albeit research on this topic has been vividly carried out before, the formal origins of the field of *enumeration complexity* can be dated to the work of Johnson, Papadimitriou and Yannakakis [JPY88b]. Certainly, algorithms in this field generally require an exponential

[1] Jonathan F. Buss and Judy Goldsmith acknowledge Samuel R. Buss for this algorithm via a personal communication reference [BG93].

runtime as the solution space of a problem instance typically is of that particular size. As a result, classifying problems problems in terms oft their overall runtime of their respective algorithms is usually abandoned. Instead, one is interested in the *delay* of the algorithm: The delay of an enumeration algorithm measures, in worst-case complexity notion, the elapsed time during outputting two consecutive solutions. Significantly to note, such algorithms must avoid printing duplicated solutions (in the whole process), that is, the same solution never is output more than once. It is worth to mention that this restriction might impose tremendous constraints on space requirements of an enumeration algorithm; in the naïve approach, all output solutions are stored in a priority queue (of possibly exponential space). The class of all problems with enumeration algorithms outputting solutions with polynomial delay (DelayP) is a central complexity class in this field. Despite the disregarded space requirements, this class widely is seen as an efficient way of enumeration in this area of research. Typical representatives of DelayP encompass the enumeration of satisfying assignments for Horn or Krom formulas [CH97], structures for first-order query problems with possibly free second-order variables and at most one existential quantifier [DS11], or cycles in graphs [RT75].

Beyond TSP, the most prominent NP-complete problem is SAT as it is the first problem which was proven to be of this particular completeness by Cook [Coo71] and, independently behind the Iron Curtain, by Levin [Lev73]. As a consequence, SAT received a wealth of theoretical attention until today and is presumably one of the most analysed and tackled problems in science. Today, there even exists an association [SAT] whose aim "*is to promote science and research, in particular with regard to the Satisfiability Problem and related areas.*" This utmost interest of researchers all over the globe brought highly optimised algorithms, known as SAT solvers today. The mentioned SAT-association even organises a SAT conference every year on which state-of-the-art SAT solvers compete in challenges. Surprisingly, these algorithms solve formulas with even hundred thousands or millions of variables [BHJ16] while brute-forcing such instances is hopeless already for hundreds of variables on basic computers. Clearly, this situation shows a vast theory-practice gap: As described before, the NP-completeness of SAT should make it impossible for the problem to be of practical relevance. Nevertheless, instances of the problem are routinely solved every single day by these SAT solvers in industry. As a result, one could reasonably question the meaningfulness of this theory. Nevertheless, the theory is still valid as it has proven that there exist hard instances of SAT which even these solvers cannot solve in an acceptable amount of time. Despite this, formula instances occurring in practice seem to embody more structure (in a similar way as explained for parametrised complexity theory). One promising approach to improve understanding the structuredness of SAT is given by the area of proof theory [Elf+16; Nor15]. Addressing this observation from a different angle, as a notable result of the "practical tractability" of SAT, many other problems are presently *translated into* SAT [BHJ17]. That means, one describes another problem instance with the help

of a propositional formula and subsequently attacks this problem with a powerful SAT solver. The computed solutions, that is, satisfying assignments, then correspond to some desired information depending on the selected problem from which one translated into SAT. As explained in the beginning of this section, one often is interested in all solutions or only the *cheap* ones (that might be generated in the beginning). Consequently, enumerating satisfying assignments is a very relevant topic of research. However, of course, efficiently outputting solutions for arbitrary SAT instances still is out of the question (as again, it would yield the collapse of P and NP).

In view of the previous remarks, we elaborate in the following on a fast way to enumerate solutions for a specific case of formulas, namely, Horn formulas. This type of formulas, named after Alfred Horn, permits a fast decision algorithm. Horn formulas consist of conjunctions of Horn *clauses* which can be written in an *implication form*. For instance, the following formulas are Horn clauses:

$$x \to y, \qquad 1 \to x, \qquad x \wedge y \to 0, \qquad x \wedge y \to z.$$

All these formulas can be rewritten using negation and disjunction:

$$\neg x \vee y, \qquad x, \qquad \neg x \vee \neg y, \qquad \neg x \vee \neg y \vee z.$$

We see that such clauses have at most one single positive variable—the remaining variables are negated. As already mentioned, such formulas can be efficiently checked for satisfiability. The idea is to search for *unit clauses*, that is, clauses of size one. For such clauses the variable has to be set to *true*. In our case this happens for x, so we set x to *true*. Then, this procedure is executed on the following modified formula:

(1.) Delete the lastly detected unit clause.

(2.) If this clause was x then delete all $\neg x$ from the formula and remove all clauses containing x.

(3.) If this clause was $\neg x$ then delete all x from the formula and remove all clauses containing $\neg x$.

We repeat this process as long as we can find unit clauses and accept if this is no longer the case (and the formula does not contain empty clauses). In the running example, we have created a new unit clause from the first Horn clause: y is set to *true*. In the next iteration, we could detect the clause z (from the last clause), however, the third Horn clause became already empty (by deleting $\neg y$ due to modification rule (2.)). Empty clauses in a formula intuitively mean that the clause became *false* and accordingly the formula is *false* under the currently evaluated assignment. As a result, the algorithm stops by saying that the input formula is not satisfiable. If the algorithm does not spot neither an empty clause nor a unit clause, then the formula is satisfiable.

Now, let us return to the particular enumeration algorithm for printing satisfying assignments of Horn formulas. It is a recursive algorithm with two parameters: A given formula F and a partial assignment A on the variables of F. In the following, let $F(A)$ denote the formula that is obtained by inserting all truth values of variables as specified by A in F (in a similar fashion as explained above). It is important to mention that F being Horn implies the same for $F(A)$. The recursive algorithm iteratively constructs assignments. We say that the assignment A is not total if not every variable in F has a specified truth value.

(1.) If A satisfies F and all variables of F have a value under A then A is printed.

(2.) If $F(A)$ is satisfiable and A is not total yet then select a variable x for which A is not defined and recursively check $(F(A \cup \{x = 0\}), A \cup \{x = 0\})$ and $(F(A \cup \{x = 1\}), A \cup \{x = 1\})$.

The first step of this algorithm clearly is in polynomial time as the evaluation of formulas under a given assignment is in P (it can even be solved by polynomial-sized circuits with logarithmic depth and bounded fan-in [Bus+92]). Step two uses the polynomial time decision algorithm sketched above and then branches recursively. The satisfiability check here is crucial to the success of the algorithm obeying a polynomial delay as "wrong" assignments immediately are discontinued in the following process.

What is the idea of the algorithm? The algorithm repeatedly reduces the instance to a smaller one and recursively builds a bounded search tree on the assignments. These two approaches, *self-reducibility* and the technique of *bounded search trees* are fundamental concepts in enumeration complexity that will also play a role in Section 4.2. Besides, these compressing steps resemble kernels from parametrised complexity theory.

Beyond the search for polynomial delay algorithms, a special type of delay has been studied: *incremental polynomial time* (IncP), that is, the i-th delay is polynomial in the input length *and* in i. While initially, the runtime is polynomial, later it eventually becomes exponential (for solution spaces of that size). A well-studied problem in this enumeration complexity class is the task of generating all maximal solutions of systems of equations modulo 2 [Kha+05]. Even today, it is not clear whether this problem can be solved with a polynomial delay. More, an open question of research is the exact relation of IncP to DelayP.

Parametrised Enumeration. The theory of parametrised enumeration allows for a fine-grained enumeration complexity analysis from the perspective of parametrised complexity. In this thesis, we establish this young field by presenting its formal foundations and first results. Prior to the research of Creignou et al. [Cre+13; Cre+15; Cre+17b; Cre+19], some preliminary work has existed. In 2002 and 2006, Fernau [Fer02] and Damaschke [Dam06] already considered a way of FPT-enumeration algorithms which we will specify later. They focussed, respectively, on cardinality minimum and inclusion minimal solutions for *hitting set* problems, and

other subset minimisation problems. Of course, the existence of such an enumeration algorithm requires only an FPT-number[2] of minimal solutions for every instance, which is quite restrictive. Nevertheless, Fernau was able to show that the problem to enumerate all minimum cardinality vertex covers is possible in FPT-time. Of course, considering the general vertex cover problem preempts the existence of an FPT enumeration algorithm as the number of distinct vertex-covers exceeds any FPT-like runtime in the worst case. This observation clearly motivates to further investigate the algorithmic possibilities for this problem. In Section 4.1, we will see how this problem can be approached to present an enumeration algorithm running with FPT-delay.

In this thesis, we will see how parametrised enumeration unites parametrised complexity theory with enumeration complexity theory, utilises and benefits from several pivotal techniques stemming from both areas. One of these paradigms will be computing the kernel of an instance (kernelisation) [Dam06; FSV13] (see Section 4.1). For this algorithmic process we will create a new method, enum-kernelisation, which characterises the parametrised enumeration complexity class DelayFPT. Furthermore, we show how self-reducibility [Sch76; KV91; CH97; Sch09] (cf. Section 4.2) together with bounded search trees can be employed to enumerate satisfying assignments for formulas of a specific SAT variant; this will yield a complete dichotomous classification for this problem where all other cases cannot be efficiently enumerated in DelayFPT. Another application in the scope of backdoor sets confirms that our definitions match.

Furthermore, we will extend the new theory of parametrised enumeration algorithms by orders, that is, we enable our theory to cope with partial orders on the solution space (which must be obeyed during the enumeration process). We exemplify this extension via kinds of graph modification problems. Finally, this section culminates in the construction of a completely generic enumeration scheme for very general modification problems which will be demonstrated in numerous ways. The last part of this thesis provides one of the first enumeration complexity results in the area of the quite young team logics which touch many other disciplines of research such as linguistics [Abr+13], biology [Abr+13], game [Bra15] and social choice theory [Shp15].

1.3 Team Logics

Expressing dependencies within formulas has a history which goes back to *branching quantifiers* by Henkin [Hen59]. Three decades later, Hintikka and Sandu [HS89] brought *independence-friendly (IF) logic* (which Tulenheimo [Tul04] investigated in the modal setting), and Hodges [Hod97a; Hod97b] considered compositional semantics for it. This type of semantics can be seen as the cornerstone of the family

[2] We use the class name FPT to abbreviate a function of the form $f(k) \cdot p(n)$, where f is a computable function, p a polynomial, k a parameter value, and n the input length. That is, we describe quantities or runtimes in terms of how the runtimes of machines for languages in FPT are defined.

of modern logics of dependence and independence by Väänänen [Vää07; Vää08]. Afterwards, different concepts of operators have been introduced and studied in the context of these logics. Among them, in 2012, Galliani analysed specific *inclusion atoms* yielding *inclusion logic* [Gal12], and Grädel together with Väänänen defined *independence logic* [GV13].

Aiming for the logical modelling of uncertainty, imperfect information, or functional (in-)dependence, Väänänen realised that usual Tarskian semantics have to be abandoned: it is futile to reason with respect to single assignments in this setting. Ingeniously, he invented *team semantics* by adopting Hodges' compositional semantics. Initially, Väänänen [Vää07] considered the setting of predicate logic (first-order logic, FO). Here, a *team* corresponds to a set of FO assignments with the same domain of variables. Essentially, such an FO team is the same as a database table where variables are its attributes and the assignments are the records. At a first glance, team semantics seems to modify the usual meaning of the "or"-connective $\vee$: The disjunction of two formulas φ, ψ is satisfied by a given team T if and only if there exist a partition $T_1 \uplus T_2 = T$ such that T_1 satisfies φ and T_2 satisfies ψ. On teams of cardinality one, this is equivalent to the classical semantics of $\vee$ as, by definition, the empty team satisfies any formula. Moreover, the dependence operator eponymous for Dependence Logic (DL) is an atomic formula of the form $=(x_1, \ldots, x_n, y)$ where the x_i (for $1 \leq i \leq n$) and y are variables. Then, such an atomic formula is satisfied by a team T if and only if all assignments that agree on the x_i also agree on y. Accordingly, the value of y is functionally determined by the x_is. Now the connection to database theory becomes more evident as dependence atoms reassemble functional dependencies in this logic.

In the following, we will give a brief glimpse into the expressive power of the two mentioned connectives, $\vee$ and $=()$, by sketching a result of Lohmann [Loh12, Thm. 4.13]. We will consider the model-checking problem in propositional dependence logic, that is, given a set of propositional assignments (the team) and a formula, asking if the team satisfies the formula. Now, the much stressed problem SAT stays NP-complete for formulas of 3CNF (conjunctive normal form with three elements per clause), that is for instance, formulas of the kind

$$\varphi := (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_4 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_4).$$

Let us recall that the three subformulas with only $\vee$ inside are called clauses. Formulas in 3CNF can be translated into a model-checking instance of propositional dependence logic as follows. On the one hand, we use a formula of the type

$$f(\varphi) := (t_1 \wedge =(v_1)) \vee (t_2 \wedge =(v_2)) \vee (t_3 \wedge =(v_3)) \vee (t_4 \wedge =(v_4)).$$

saying that there exists a split of the team T into four subteams $T_1, \ldots, T_4$ (for each variable of φ one, required by the disjunctions) and v_j is constant in T_j. Note that

some of these subteams might be empty. Intuitively, we will use in the following the combination of v_i and t_i to express the *parity of the variable* x_i in a clause. That is, we want to mimic whether x_i or $\neg x_i$ is in that very clause. Before we can completely understand the interplay of these variables, we need to introduce the team $T := \{c_1, \ldots, c_3\}$:

$$c_i(v_j) = \begin{cases} 1, & \text{if } x_j \text{ is in clause } i \\ 0, & \text{otherwise,} \end{cases} \qquad c_i(t_j) = \begin{cases} 1, & \text{if } x_j \text{ or } \neg x_j \text{ is in clause } i \\ 0, & \text{otherwise.} \end{cases}$$

Informally, c_i encodes the i-th clause of φ. Let us consider the construction for the propositional formula from above:

	v_1	v_2	v_3	v_4	t_1	t_2	t_3	t_4
c_1	1	0	1	0	1	1	1	0
c_2	0	1	0	0	1	1	0	1
c_3	1	0	0	1	1	1	0	1

Let us turn back to the aforementioned parity property. In order to satisfy $f(\varphi)$ by the team T, we need to divide T such that in a subteam T_i we have no two c, c' with opposite v_i-values. Accordingly, there are no two assignments in T_i which correspond to clauses in φ such that one contains x_i and the other $\neg x_i$. More, all c-entries in T_i have the *same* v_i (and thereby either all positive or negative x_i in the corresponding clause) and t_i is set to 1.

How is 'satisfiability of φ' related to 'T fulfilling $f(\varphi)$'? A satisfying assignment A of φ, for instance, is $A(x_1) = 1 = A(x_2)$ and the other two variables are set to 0. In the following, we will explain how T can be split into subteams such that each subteam satisfies the corresponding subformula of $f(\varphi)$. Assign $c_1, c_3 \in T_1$ and $c_2 \in T_2$. The subteam T_1, for variable x_1, satisfies t_1 and v_1 is constant 1 here as x_1 occurs in the first and third clause positively. Likewise T_2 satisfies t_2 and v_2 again is constant 1. Principally, T_i contains all "clauses" c_j which are satisfied by variable x_i with a constant (0 or 1) truth value guaranteed by $=(v_i)$. Now, we have seen how a satisfying assignment leads to a satisfying partition of the team. For the other direction, that from a satisfying splitting one can construct a satisfying assignment for φ, consider the following argument. We show how splitting into n subteams (for n variables of φ) allows to construct a satisfying assignment. One merely needs to investigate the teams T_i and detect to which constant value the respective v_i is mapped. If the value is 0 then one sets $A(x_i) = 0$ and otherwise one maps $A(x_i) = 1$. From this it follows that the formula is satisfied as all "clauses" c are satisfied and not distributed on the subteams in a contradictory fashion.

The discussed example shows that a dependence atom of arity one for each variable in combination with linear many splitting disjunctions and conjunctions

already suffices to express an NP-complete problem. By this observation, one may guess that richer logics in this area might encode highly complex problems.

As we have mentioned in the beginning of this section, history showed how many different logics can be adapted to team semantics. In this thesis, we consider a very weak version of propositional team logic and study, for the first time, the parametrised complexity of enumeration in a team logic. This research can be seen as an initial step in the currently unknown landscape of enumeration complexity of the team logics of dependence and independence.

1.4 Results

In this thesis, we lay out the foundations of the framework of parametrised enumeration. This is an important first step which has to be conducted very carefully. Subsequently, we thoroughly examine this heart of the matter and ensure the definitions are adequate. We then approve the introduced classes and consider the landscape of these classes (Chapter 3). Here, we particularly work with the class of *incremental FPT enumeration* (IncFPT), and connect the area of parametrised numeration to its classical counterpart (Section 3.2). In particular, newly introduced parametrised function classes are utilised to prove this touch point of the two areas of research. In the setting of parametrised enumeration, we study two variants: unordered (Chapter 4) and ordered enumeration (Chapter 5). For unordered enumeration, we investigate how two crucial techniques, kernels (Section 4.1) and self-reducibility (Section 4.2), can be transferred to this new framework. These results again underline that our definitions are sensible. We close this part by presenting a dichotomy theorem (Section 4.3). Turning towards ordered enumeration, we show the significance of graph modification problems [BHL14] in this context (Section 5.1). Such problems, relevant in copious areas of research, encompass the modification of a graph by deletion or introduction of edges, respectively, removing vertices to transform the graph into another obeying a specific property (for example, being *triangular*, that is, all its induced cycles are triangles ). In this context, we will see how a specific characterisation known from parametrised complexity theory plays a crucial role in allowing efficient enumeration in this setting. Afterwards, we fundamentally generalise this technique to obtain a parametrised enumeration scheme for arbitrary modification problems (Section 5.2). Finally, we study parametrised enumeration of *Poor Man's Propositional Dependence Logic* (Chapter 6) and present the first enumeration results for a team logic.

1.5 Publications

Chapter 3 discusses new and unpublished insights into the classes IncFPT and OutputFPT. Furthermore, it presents the first connection to classical enumeration complexity theory and finishes with some remarks on a cardinality version of the SAT problem. In this section, we define the parametrised version of the class of

nondeterministic multivalued functions with values that are polynomially verifiable and guaranteed to exist, TF(NP), known from Megiddo and Papadimitriou [MP91]. We show that a collapse of TF(NP) to FP is equivalent to a collapse of OutputFPT to IncFPT. There exists only a technical report on these results, which can be found at the *Computing Research Repository (CoRR)* [Mei18]. The results in Chapter 4 and Section 4.3 have been published at the *38th International Symposium on Mathematical Foundations of Computer Science* in 2013 [Cre+13], and in the Springer journal *Theory of Computing Systems* [Cre+17b]. Parts of Chapter 5 have been published at the *9th International Conference on Language and Automata Theory and Applications* in 2015 [Cre+15], and in the MDPI journal *Algorithms* in 2019 [Cre+19]. The results in Chapter 6 advance the classical enumeration results of a publication at the *Tenth International Symposium on Foundations of Information and Knowledge Systems, 2018* [MR18] and of a technical report, which can be found at the *Computing Research Repository (CoRR)* [MR17], to parametrised enumeration complexity.

Preliminaries

2

> *The whole of science is nothing more than a refinement of everyday thinking.*
>
> **— Albert Einstein**

In this chapter we introduce the required mathematical tools and notions for this thesis. We assume familiarity with basic notions in logics and complexity theory. If a deeper introduction is needed, we refer the reader to the textbooks of Mendelson [Men87] and Sipser [Sip12].

The truth values *true* and *false* are denoted with the symbols $\top$ and $\bot$. Propositional variables are usually denoted with x, y, z and formulas with greek letters φ, ψ. If φ is a formula, x and t are variables, then $\varphi(t/x)$ is φ where each occurrence of x is replaced by t.

2.1 Complexity Theory

Most algorithms constructed in this thesis are using exponential space in the input size while outputting the solutions of a given instance. As Turing machines cannot access specific bits of exponentially sized data in polynomial time, one commonly uses the RAM model as the machinery of choice; see the work of Creignou et al. [Cre+17a]. Usually, polynomially restricted RAMs (RAMs where each register content is polynomially bounded w.r.t. the input size) suffice for our purposes.

Besides the standard complexity classes P and NP, we will make use of the classes for (non)deterministic doubly exponential time, that is, $\mathsf{EE} := \mathsf{DTIME}(2^{2^n})$ and $\mathsf{NEE} := \mathsf{NTIME}(2^{2^n})$. In this context, $\mathsf{DTIME}(f)$, respectively, $\mathsf{NTIME}(f)$ refers to deterministic, respectively, nondeterministic RAMs running in time $O(f)$ for some runtime function $f\colon \mathbb{N} \to \mathbb{N}$.

2.2 Parametrised Complexity Theory

In this section, we will present a brief introduction into the field of parametrised complexity theory. For a deeper introduction we kindly refer the reader to the textbooks of Flum and Grohe [FG06], Niedermeyer [Nie06], or Downey and Fellows [DF13].

Consider a decision problem $Q \subseteq \Sigma^*$ over some alphabet Σ. Then $L \subseteq Q \times \Sigma^*$ is called a *parametrised language* (or problem). Given an instance $\langle x, k\rangle \in Q \times \Sigma^*$, we call k the *parameter (of* L*)*. Often, a polynomial time computable function $\kappa\colon \Sigma^* \to \Sigma^*$, the *parametrisation*, is used to directly address this parameter. In such a case, one also writes (Q, κ) to denote the parametrised problem. In most cases,

the codomain of the parameter is a natural number and as a result, L is a subset of $\Sigma^* \times \mathbb{N}$ and then $\kappa\colon \Sigma^* \to \mathbb{N}$ also is defined via the set of natural numbers.

Definition 2.1 (Fixed-parameter tractable)**.**
Let (Q, κ) *be a parametrised problem over some alphabet* Σ*. If there exists a deterministic algorithm* A *and a computable function* $f\colon \mathbb{N} \to \mathbb{N}$ *such that for all* $x \in \Sigma^*$

- A *accepts* x *if and only if* $x \in Q$*, and*
- A *has a runtime of* $O(f(\kappa(x)) \cdot |x|^{O(1)})$*,*

then A *is an* fpt-algorithm for (Q, κ) *and* (Q, κ) *is* fixed-parameter tractable *(or short, in the complexity class* FPT*)*.

In Section 4.3, we will require an appropriate notion of reductions in this setting. The following definition serves for this purpose.

Definition 2.2 (fpt-reduction)**.**
Let (Q, κ) *and* (P, λ) *be two parametrised problems over alphabets* Σ *and* Γ*. Then a function* $f\colon \Sigma^* \to \Gamma^*$ *is an* fpt-reduction from (Q, κ) to (P, λ)*, in symbols* $(Q, \kappa) \leq_{\mathbf{fpt}} (P, \lambda)$*, if* f *can be computed in* FPT *time w.r.t.* κ*, and there exists a computable function* $h\colon \mathbb{N} \to \mathbb{N}$ *such that*

- $x \in P$ *if and only if* $f(x) \in Q$*, and*
- $\lambda(f(x)) \leq h(\kappa(x))$.

Analogously to showing usual NP-hardness with respect to $\leq^{\mathbf{p}}_{\mathbf{m}}$-reductions, in parametrised complexity theory one aims for lower bounds for the complexity class W[1] (which we define presently) with respect to $\leq_{\mathbf{fpt}}$-reductions.

Definition 2.3.
The class W[1] *is the class of parametrised problems* (Q, κ) *which can be* $\leq_{\mathbf{fpt}}$*-reduced to the following problem*

Problem:	Short Single-Tape Turing Machine Halting Problem
Input:	Single-tape NTM M, integer k.
Parameter:	The integer k.
Question:	Does M accept the empty string in $\leq k$ steps?

It is worth to mention that the W-symbol for the complexity class stems from a different characterisation of this complexity class which is the *weft* of a circuit. A circuit has weft $i \in \mathbb{N}$ if on every directed path from input to output gates there exist at most i gates of fan-in > 2. The class W[1] is then the class of all circuits of weft 1. Similarly, a hierarchy of W[]-classes arises. It is known that $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \cdots \subseteq \mathsf{W}[P]$, where W[P] is the set of languages which can be decided by a NTM in time $O(f(\kappa(x)) \cdot |x|^{O(1)})$ with at most $O(\kappa(x) \cdot \log|x|)$ nondeterministic steps. One can see that W[P] feels like FPT with increased nondeterministic power, yet it is only known that $\mathsf{FPT} \subseteq \mathsf{W}[P]$; for instance, $\mathsf{FPT} = \mathsf{W}[P]$ implies $P = \mathsf{W}[P]$.

A way to "parametrise" a classical and *robust* complexity class is given by Flum and Grohe [FG06] and is utilised in the next definition.

Definition 2.4 (para-NP, [FG06, Def. 2.10])**.**
Let (Q, κ) *with* $Q \subseteq \Sigma^*$ *be a parametrised problem over some alphabet* Σ*. We have* $(Q, \kappa) \in$ para-NP *if there exists a computable function* $f\colon \mathbb{N} \to \mathbb{N}$ *and a nondeterministic algorithm* N *such that for all* $x \in \Sigma^*$ N *correctly decides* $x \in Q$ *in at most* $f(\kappa(x)) \cdot p(|x|)$ *steps, where* p *is a polynomial.*

Flum and Grohe present also an alternative characterisation of the class para-NP via all problems "*that are in* NP *after precomputation on the parameter*".

Proposition 2.5 ([FG06, Prop. 2.12])**.**
Let (Q, κ) *be a parametrised problem over some alphabet* Σ*. We have* $(Q, \kappa) \in$ para-NP *if there exists a computable function* $\pi\colon \mathbb{N} \to \Sigma^*$ *and a problem* $Q' \subseteq \Sigma^* \times \Sigma^*$ *such that* $Q' \in$ NP *and the following is true: for all instances* $x \in \Sigma^*$ *we have that* $x \in Q$ *if and only if* $(x, \pi(\kappa(x))) \in Q'$*.*

According to the well-known characterisation of the complexity class NP via a verifier language, one can easily deduce the following corollary.

Corollary 2.6.
Let (Q, κ) *be a parametrised problem over some alphabet* Σ *and* p *some polynomial. We have* $(Q, \kappa) \in$ para-NP *if there exists a computable function* $\pi\colon \mathbb{N} \to \Sigma^*$ *and a problem* $Q' \subseteq \Sigma^* \times \Sigma^* \times \Sigma^*$ *such that* $Q' \in$ P *and the following is true: for all instances* $x \in \Sigma^*$ *we have that* $x \in Q$ *if and only if there exists a* y *such that* $|y| \leq p(|x|)$ *and* $(x, \pi(\kappa(x)), y) \in Q'$*.*

2.3 Enumeration

Typically, the set of solutions for a given problem instance is much larger than the input size, namely, it is exponentially larger. As a consequence, considering polynomial time algorithms is not meaningful while talking about an efficient performance. That being so, one inspects the uniformity of the flow of output solutions of these algorithms rather their total runtime. In view of this, one measures the delay between two consecutive outputs. Johnson et al. laid the cornerstone of this intuition in a seminal paper [JPY88b] and introduced the necessary tools and complexity notions. Creignou, Olive, and Schmidt [COS11; Sch09] present recent notions in this framework, which we aim to follow.

Definition 2.7 (Enumeration problem)**.**
An enumeration problem *(over an alphabet* Σ*) is a tuple* $E = (Q, \mathrm{Sol})$*, where*

(1.) $Q \subseteq \Sigma^*$ *is the set of instances (recognisable in polynomial time),*

(2.) Sol *is a computable function such that for all* $x \in \Sigma^*$ *the set* $\mathrm{Sol}(x)$ *is the finite set of solutions of* x*,*

(3.) $\{(x, y) \mid y \in \mathrm{Sol}(x)\} \in$ P*, and*

(4.) there exists a polynomial such that for all $x \in Q$ *and* $y \in \mathrm{Sol}(x)$ *we have* $|y| \leq p(|x|)$.

Then an *enumeration algorithm* $\mathcal{A}$ for the enumeration problem $E = (Q, \mathrm{Sol})$ is a deterministic algorithm, which on the input x of E, outputs exactly the elements of $\mathrm{Sol}(x)$ without duplicates, and which terminates after a finite number of steps on every input.

As mentioned before, the overall runtime of enumeration algorithms often is exponential (or even worse) in the input size due to the vast number of possible solutions. As a result, one studies the time elapsed between to consecutive output solution. If this time can be described generally over all output solutions then we talk about the *delay* as the following definition states.

Definition 2.8 (Delay)**.**
Let $E = (Q, \mathrm{Sol})$ *be an enumeration problem and* $\mathcal{A}$ *be an enumeration algorithm for* E. *For* $x \in Q$ *we define the* i*-th delay of* $\mathcal{A}$ *as the time between outputting the* i*-th and* $(i+1)$*-st solution in* $\mathrm{Sol}(x)$. *Furthermore, we set the* 0*-th delay to be the* precomputation phase *which is the time from the start of the computation to the first output statement. Analogously, the* n*-th delay, for* $n = |\mathrm{Sol}(x)|$*, is the* postcomputation phase *which is the time needed after the last output statement until* $\mathcal{A}$ *terminates.*

Subsequently, we will use the notion of delay to state the central enumeration complexity classes.

Definition 2.9.
Let $E = (Q, \mathrm{Sol})$ *be an enumeration problem (EP) and* $\mathcal{A}$ *be an enumeration algorithm for* E. *Then* $\mathcal{A}$ *is*

(1.) an P-enum*-algorithm if and only if there exists a polynomial* p *such that for all* $x \in Q$*, algorithm* $\mathcal{A}$ *outputs* $\mathrm{Sol}(x)$ *in time* $O(p(|x|))$.

(2.) a DelayP*-algorithm if and only if there exists a polynomial* p *such that for all* $x \in Q$*, algorithm* $\mathcal{A}$ *outputs* $\mathrm{Sol}(x)$ *with delay* $O(p(|x|))$.

(3.) an IncP*-algorithm if and only if there exists a polynomial* p *such that for all* $x \in Q$*, algorithm* $\mathcal{A}$ *outputs the* $\mathrm{Sol}(x)$ *with the* i*-th delay of* $O(p(|x|, i))$ *(for every* $0 \leq i \leq |\mathrm{Sol}(x)|$*).*

(4.) a $\mathrm{CapIncP}_a$*-algorithm if and only if there exists a polynomial* p *such that for all* $x \in Q$*, algorithm* $\mathcal{A}$ *outputs* i *elements of* $\mathrm{Sol}(x)$ *in time* $O(p(|x|, i^a))$ *(for every* $0 \leq i \leq |\mathrm{Sol}(x)|$.

(5.) a OutputP*-algorithm if and only if there exists a polynomial* p *such that for all* $x \in Q$*, algorithm* $\mathcal{A}$ *outputs* $\mathrm{Sol}(x)$ *in time* $O(p(|x|, |\mathrm{Sol}(x)|))$.

Accordingly, we say that E *is in* P-enum*/*DelayP*/*IncP*/*$\mathrm{CapIncP}_a$*/*OutputP *if* E *admits an* P-enum*-/*DelayP*-/*IncP*-/*$\mathrm{CapIncP}_a$*-/*OutputP*-algorithm. Finally, we define* $\mathrm{CapIncP} := \bigcup_{a \in \mathbb{N}} \mathrm{CapIncP}_a$.

Algorithm 2.1: Enumerate all satisfying assignments of a given Krom-formula

Input: A Krom-formula ϕ, assignment θ
Output: All satisfying assignments of ϕ.

```
1 if φ[θ] ≡ 1 and Vars(θ) = Vars(φ) then output θ
2 if φ[θ] is satisfiable and Vars(θ) ≠ Vars(φ) then
3 |   choose x ∈ Vars(φ[θ])
4 |   EnumKrom (φ, θ ∪ {x = 0})
5 |   EnumKrom (φ, θ ∪ {x = 1})
```

Note that in the diploma thesis of Schmidt [Sch09, Sect. 3.1] the class P-enum is called TotalP. We avoid this name to prevent possible confusion in the following section as well as with the work of Capelli and Strozecki [CS17]. Also, we want to point out that Capelli and Strozecki use the definition of CapIncP for IncP (and use the name "UsualIncP" for IncP instead). This definitions requires an enumeration algorithm to enumerate $m \leq \mathrm{Sol}(x)$ solutions in time $O(m \cdot |x|)$. They prove that the notions of CapIncP and IncP are equivalent up to an exponential space requirement when using a structured delay. So generally, without any space restrictions, the following result applies.

Proposition 2.10 (Corollary 13 in [CS17])**.**
CapIncP = IncP.

Example 2.11. *Consider the task of enumerating all satisfying assignments of a given propositional Krom-formula* ENUM-KROM*. It is well-known that satisfiability of propositional Krom-formulas can be decided in polynomial time by merely searching for cycles containing variables and their negation in the underlying implication graph. Using this observation, the enumeration algorithm depicted in Algorithm 2.1 has polynomial delay and correctly enumerates all satisfying assignments of a given Krom formula after the initial call* $\texttt{EnumKrom}(\phi, \emptyset)$.

The correctness can be similarly proven as for the algorithm in the proof of Theorem 4.11.

2.4 Parametrised Enumeration

After we have now introduced the basic principles of parametrised complexity theory and enumeration complexity theory, following the approach of Johnson et al. [JPY88b], we will introduce the corresponding parametrised versions of these previously introduced notions.

Definition 2.12.
A parametrised enumeration problem *(PEP) over a finite alphabet* Σ *is a triple* $E = (Q, \kappa, \mathrm{Sol})$ *such that*

(1.) $Q \subseteq \Sigma^*$ *is the set of instances (recognisable in polynomial time),*

(2.) κ *is a parametrisation of* Σ^**, that is,* $\kappa\colon \Sigma^* \to \mathbb{N}$ *is a polynomial time computable function,*

(3.) $\mathrm{Sol}\colon \Sigma^* \to \mathcal{P}(\Sigma^*)$ *is a computable function such that for all* $x \in \Sigma^*$, $\mathrm{Sol}(x)$ *is a finite set and* $\mathrm{Sol}(x) \neq \emptyset$ *if and only if* $x \in Q$,

(4.) $\{\langle x, y\rangle \mid y \in \mathrm{Sol}(x)\} \in P$, *and*

(5.) there exists a polynomial p *such that for all* $x \in Q$ *and* $y \in \mathrm{Sol}(x)$ *we have* $|y| \leq p(|x|)$.

Furthermore, we use the shorthand $\mathcal{S} = \bigcup_{x \in I} \mathrm{Sol}(x)$ to refer to the set of solutions for every possible instance. If $E = (Q, \kappa, \mathrm{Sol})$ is a parametrised enumeration problem over the alphabet Σ, then we call strings $x \in \Sigma^*$ *instances of* E, the number $\kappa(x)$ the corresponding parameter, and $\mathrm{Sol}(x)$ the *set of solutions of* x. Besides, the definitions of enumeration algorithms and delays are easily adjusted to the setting of parametrised enumeration problems.

Observe that the following defined classes are in complete analogy to the non-parametrised case from the previous section.

Definition 2.13.
Let $E = (Q, \kappa, \mathrm{Sol})$ *be a parametrised enumeration problem and* $\mathcal{A}$ *an enumeration algorithm for* E. *Then the algorithm* $\mathcal{A}$ *is*

(1.) an FPT*-enumeration algorithm if there exist a computable function* $t\colon \mathbb{N} \to \mathbb{N}$ *and a polynomial* p *such that for every instance* $x \in \Sigma^*$, $\mathcal{A}$ *outputs all solutions of* $\mathrm{Sol}(x)$ *in time at most* $t(\kappa(x)) \cdot p(|x|)$,

(2.) a DelayFPT*-algorithm if there exist a computable function* $t\colon \mathbb{N} \to \mathbb{N}$ *and a polynomial* p *such that for every* $x \in \Sigma^*$, $\mathcal{A}$ *outputs all solutions of* $\mathrm{Sol}(x)$ *with delay of at most* $t(\kappa(x)) \cdot p(|x|)$,

(3.) an IncFPT*-algorithm if there exist a computable function* $t\colon \mathbb{N} \to \mathbb{N}$ *and a polynomial* p *such that for every* $x \in \Sigma^*$, $\mathcal{A}$ *outputs all solutions of* $\mathrm{Sol}(x)$ *and the* i*-th delay is at most* $t(\kappa(x)) \cdot p(|x| + i)$, *and*

(4.) an OutputFPT*-algorithm if there exist a computable function* $t\colon \mathbb{N} \to \mathbb{N}$ *and a polynomial* p *such that for every instance* $x \in \Sigma^*$, $\mathcal{A}$ *outputs all solutions of* $\mathrm{Sol}(x)$ *in time at most* $t(\kappa(x)) \cdot p(|x| + |\mathrm{Sol}(x)|)$.

Note that before, the notion of TotalFPT has been used for the class of FPT-enumerable problems [Cre+13]. We avoid this name as it causes confusion with respect to an enumeration class TotalP [Sch09, Sect. 3.1] which takes into account not only the size of the input but also the number of solutions. We call this class OutputP instead, and accordingly, it is the non-parametrised analogue of the above class OutputFPT. Now we group these different kinds of algorithms in complexity classes.

Definition 2.14 (Parametrised Enumeration Complexity Classes)**.**
The class FPT-enum*/*DelayFPT*/*IncFPT*/*OutputFPT *is the class of all parametrised enumeration problems that admit a* FPT*-enumeration/*DelayFPT*-/*IncFPT*-/*OutputFPT*-algorithm.*

In 2002, respectively, 2006, Fernau [Fer02] and Damaschke [Dam06] already considered the notion of FPT-enumeration algorithms from Definition 2.13 (1.). They focussed, respectively, in cardinality minimum and inclusion minimal solutions for problems such as VERTEX-COVER, HITTINGSET, and other subset minimisation problems. Of course, the existence of such an enumeration algorithm requires that for every instance x the number of minimal solutions is bounded by $f(\kappa(x)) \cdot p(|x|)$, which is quite restrictive. Nevertheless, Fernau was able to show that the problem MINIMUM-VERTEX-COVER (where we are only interested in vertex covers of minimum cardinality) is FPT-enumerable. Nonetheless, by the just given cardinality constraint, ENUM-VERTEX-COVER cannot be in FPT-enum. In Section 4.1 we will show that ENUM-VERTEX-COVER is in DelayFPT.

In view of this, we propose that DelayFPT should be regarded as the good notion of tractability for parametrised enumeration complexity.

Note that, due to Flum and Grohe [FG06, Prop. 1.34] the class FPT can be characterised via runtimes of the form either $f(\kappa(x)) \cdot p(|x|)$ or $f(\kappa(x)) + p(|x|)$ (as $a \cdot b \leq a^2 + b^2$, for all $a, b \in \mathbb{N}$). Accordingly, this applies also to the introduced classes DelayFPT, IncFPT, and OutputFPT.

2.5 Orders

In this section we introduce an ordering relation on the solution set for a given parametrised enumeration problem. This relation should be a preorder, that is, a reflexive, transitive, binary relation.

Definition 2.15.
A parametrised enumeration problem with ordering $E = (I, \kappa, \mathrm{Sol}, \preceq)$ *is a parametrised enumeration problem, where $\preceq$ is a quasiorder (or* preorder*) on the set of all solutions $\mathcal{S}$.*

Example 2.16. *The reachability relation between vertices of a given directed graph is a quasiorder. The divisibility relation on the set of integers is a quasiorder as well.*

Additionally, we have to extend the definition of enumeration algorithms such that the presented output has to obey the given quasiorder.

Definition 2.17 (Ordered Enumeration Algorithm)**.**
Let $E = (I, \kappa, \mathrm{Sol}, \preceq)$ be a parametrised enumeration problem with ordering. Then an algorithm $\mathcal{A}$ is an ordered enumeration algorithm for E *if it is an enumeration algorithm and additionally for every $x \in I$ and $y, z \in \mathrm{Sol}(x)$, if $y \preceq z$ and $z \not\preceq y$ then $\mathcal{A}(x)$ outputs solution y before solution z.*

Again, the definition of delay and the corresponding complexity classes are also lifted to ordered enumeration algorithms.

Later, in Chapter 5, some of our enumeration algorithms will make use of the concept of *priority queues* to enumerate all solutions in the correct order and to avoid duplicates. This technique builds on the approach of Johnson et al. [JPY88b].

A priority queue Q stores a potentially exponential number of elements. Now let x be an instance. Then, the *insert operation* of Q requires $O(|x| \cdot \log |\mathrm{Sol}(x)|)$ time. The *extract minimum operation* requires $O(|x| \cdot \log |\mathrm{Sol}(x)|)$ time, too. Importantly, nonetheless, the computation of the order between two elements takes at most $O(|x|)$ time. As pointed out by Johnson et al. the described queue can be implemented with the help of standard balanced tree schemes [AFL04].

3 Enumeration Complexity Landscape

> *One of the most interesting things about theoretical computer science is how often the universe plays tricks on us. Our intuition may say things should be one way, yet the universe is being playful, and something quite different holds.*
>
> **— Lane A. Hemaspaandra**
> [Hem16]

In this chapter we will reconsider the precision of the definitions from the previous chapter. Let us begin with the class DelayFPT. Surprisingly, the FPT delay for the class DelayFPT need not be continuously maintained. It can be moved into the FPT-precomputation phase and then is followed by enumeration with polynomial delay. Formally, the class StrictDelayFPT is the class of all parametrised enumerations problems that admit a DelayP enumeration algorithm with precomputation phase of FPT.

Theorem 3.1.
DelayFPT = StrictDelayFPT.

Proof. Let $\mathcal{A}$ be a DelayFPT-enumeration algorithm for $(Q, \kappa, \mathrm{Sol})$. Choose a computable function t and a polynomial p such that on input x the algorithm $\mathcal{A}$ outputs all elements of $\mathrm{Sol}(x)$ with delay of at most $t(\kappa(x)) \cdot p(|x|)$. Define $f\colon \mathbb{N} \to \mathbb{N}$ by

$$f(m) = \max_{y \in \Sigma^*, |y| \leq m} \{\, s \mid s \text{ runtime of } \mathcal{A} \text{ on input } y \,\}.$$

Then the following algorithm $\mathcal{B}$ is an enumeration algorithm for $(Q, \kappa, \mathrm{Sol})$ with polynomial delay (and FPT-precomputation phase). On input x it checks (as part of the precomputation) whether $t(\kappa(x)) \leq |x|$. If so, then $\mathcal{B}$ simulates $\mathcal{A}$ on x (delay $\leq |x| \cdot p(|x|)$). If $t(\kappa(x)) > |x|$, then $\mathcal{B}$ (as part of the precomputation) just simulates $\mathcal{A}$ on x thereby computing $\mathrm{Sol}(x)$ in time bounded by $f(t(\kappa(x)))$. Then $\mathcal{B}$ outputs the elements of $\mathrm{Sol}(x)$ step by step. □

Subsequently, we examine the classes IncFPT and OutputFPT. These have been defined for enumeration algorithms whose runtime of the i-th delay is $O(t(\kappa(x)) \cdot p(|x| + i))$, respectively, the overall runtime is $O(t(\kappa(x)) \cdot p(|x| + |\mathrm{Sol}(x)|))$. Note that, due to Flum and Grohe [FG06, Prop. 1.34] the class FPT can be characterised via runtimes of the form either $f(\kappa(x)) \cdot p(|x|)$ or $f(\kappa(x)) + p(|x|)$ (as $a + b \leq a^2 + b^2$,

for all $a, b \in \mathbb{N}$). Accordingly, this applies also to the introduced classes DelayFPT, IncFPT, and OutputFPT.

Similarly as in the classical setting, dropping the verifier-condition for enumeration problems, that is $\{\langle x, y\rangle \mid y \in \mathrm{Sol}(x)\} \in \mathsf{P}$ (Definition 2.12 (4.)), allows for separating incremental FPT time from output FPT time. Denote this change of the two classes IncFPT and OutputFPT via $\mathsf{IncFPT}_{\mathsf{nv}}$ and $\mathsf{OutputFPT}_{\mathsf{nv}}$ ('nv' abbreviates 'no verifier-condition').

Theorem 3.2.
$\mathsf{IncFPT}_{\mathsf{nv}} \subsetneq \mathsf{OutputFPT}_{\mathsf{nv}}$.

Proof. We will follow the ideas from Schmidt [Sch09, Prop. 3.10]. Let $Q \subseteq \Sigma^*$ and $H = (Q, \kappa)$ be an XP-complete problem w.r.t. $\leq_{\mathbf{fpt}}$-reduction and a runtime bound of $O(|x|^{f(\kappa(x))})$. Now let χ_H be the characteristic function of H. The enumeration problem under investigation is

Problem:	ENUM-ALL-STRINGS(H)
Input:	$x \in \Sigma^*$, $\kappa(x) \in \mathbb{N}$..
Parameter:	$\kappa(x)$.
Output:	The set of strings $w \in \{a_1, a_2, \ldots, a_{|x|}\}^{f(\kappa(x))} \circ \chi_H(x)$.

To solve ENUM-ALL-STRINGS(H) $\in \mathsf{OutputFPT}_{\mathsf{nv}}$, we have time of at most $t(\kappa(x)) \cdot p(|x| + |\mathrm{Sol}(x)|)$ for input $x \in \Sigma^*$, where t is a function and p is a polynomial. The size of $|\mathrm{Sol}(x)|$ is $|x|^{f(\kappa(x))}$. The computation of $\chi_H(x)$ requires $|x|^{f(\kappa(x))}$ time. After $\chi_H(x)$ has been determined we can generate a solution in time $O(f(\kappa(x)))$. As a result, we compute $\mathrm{Sol}(x)$ in

$$O(|x|^{f(\kappa(x))} + f(\kappa(x)) \cdot |\mathrm{Sol}(x)|) = O(f(\kappa(x)) \cdot |\mathrm{Sol}(x)|),$$

as $|\mathrm{Sol}(x)| = |x|^{f(\kappa(x))}$. However, ENUM-ALL-STRINGS(H) $\in \mathsf{IncFPT}_{\mathsf{nv}}$ implies the computation of $\chi_H(x)$ in FPT which contradicts $\mathsf{FPT} \subsetneq \mathsf{XP}$ ([FG06, Cor. 2.26]). Accordingly, we deduce that ENUM-ALL-STRINGS(H) $\notin \mathsf{IncFPT}_{\mathsf{nv}}$. □

Clearly, one strives for a similar result of the unrestricted classes IncFPT and OutputFPT, which would imply that $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$ (following from results of Section 3.2). Unfortunately, it is not clear how to modify the above proof in that way as it might even imply the collapse of EXP to NP.

Nevertheless, separating (enumeration) complexity classes is an important line of research, in particular, as there exists no satisfying definition of reductions in this area. Capelli and Strozecki [CS17] explain that some kinds of reductions where studied in the process (for instance, parsimonious [CS17, Def. 3], or in the setting of counting problems [DHK05]). However, reductions for enumeration problems as in Definition 2.7 or 2.12 yielding complete-problems for enumeration complexity classes are not known. Creignou et al. [Cre+17a] overcome this defect and introduce

different variations of enumeration reductions within a framework of polynomial hierarchy styled enumeration classes. With this new framework it is possible to directly state completeness-results. Before, lower bounds were conditionally related to complexity theoretic assumptions (for instance, of the form "unless $P = NP$").

3.1 Incremental FPT

The previous observations raise the question on how DelayFPT relates to IncFPT. In the classical enumeration world, this question is answered by Capelli and Strozecki [CS17, Prop. 16] for the capped version of incremental polynomial time: $\mathsf{DelayP} \subsetneq \mathsf{CapIncP}$ is true, only for linear total time and polynomial space it has not been answered yet. This is the question how DelayP with polynomial space relates to $\mathsf{CapIncP}_1$ with polynomial space [CS17, Open Problem 1, p.10]. In the course of this chapter, we will realise that the relationship between the classical and the parametrised world is very close. Capelli and Strozecki approach the separation mentioned above through the classes $\mathsf{CapIncP}_a$ and prove a strict hierarchy of these classes. We lift this to the parametrised setting.

This remark perfectly contrasts to what we will observe in Chapter 5. There, we will notice that outputting solutions ordered by their size seems to require exponential space in case one aims for DelayFPT. Also, in Chapter 6, we will observe how a DelayFPT algorithm with exponential space is transformed into an IncFPT algorithm with polynomial space. Capelli and Strozecki approach their question through refining the notion of incremental polynomial time by emphasising on the exponent of the polynomial in the outputted solutions. More precisely, they define classes IncP_a (for $a \in \mathbb{N}$) as the set of enumeration problems solvable by an algorithm whose i-th delay requires time $O(i^a \cdot p(n))$ where p is a polynomial and n is the input length and they prove a strict hierarchy of these classes. Subsequently, we will investigate whether one can observe the same in the parametrised setting.

Definition 3.3 (Sliced Versions of Incremental FPT, extending Def. 2.13)**.**

3'. an $\mathsf{CapIncFPT}_a$*-algorithm (for* $a \in \mathbb{N}$*) if there exists a computable function* $t\colon \mathbb{N} \to \mathbb{N}$ *and a polynomial* p *such that for every* $x \in \Sigma^*$*,* $\mathcal{A}$ *outputs* i *elements of* $\mathrm{Sol}(x)$ *in time* $t(\kappa(x)) \cdot i^a \cdot p(|x|)$ *(for every* $0 \leq i \leq |\mathrm{Sol}(x)|$*).*

3". an IncFPT_a*-algorithm (for* $a \in \mathbb{N}$*) if there exists a computable function* $t\colon \mathbb{N} \to \mathbb{N}$ *and a polynomial* p *such that for every* $x \in \Sigma^*$*,* $\mathcal{A}$ *outputs all solutions of* $\mathrm{Sol}(x)$ *and the* i*-th delay is at most* $t(\kappa(x)) \cdot i^a \cdot p(|x|)$*.*

Similarly, we define a hierarchy of classes $\mathsf{CapIncFPT}_a$ *for every* $a \in \mathbb{N}$ *which consist of problems that admit an* $\mathsf{CapIncFPT}_a$*-algorithm. Moreover,* $\mathsf{CapIncFPT} := \bigcup_{a\in\mathbb{N}} \mathsf{CapIncFPT}_a$.

Clearly, $\bigcup_{a\in\mathbb{N}} \mathsf{IncFPT}_a = \mathsf{IncFPT}$ and $\mathsf{IncFPT}_0 = \mathsf{DelayFPT}$ by Definition 2.13 as the i-th delay then merely is $t(\kappa(x)) \cdot p(|x|)$, as $i^0 = 1$.

Agreeing with Capelli and Strozecki [CS17, Sect. 3], it seems very reasonable to see the difference of IncFPT_1 and DelayFPT anchored in the exponential sized priority queue. The price of a "regular" (that is, polynomial) delay is paid by requiring exponential space. Though, relaxing this statement shows that the equivalence of incremental FPT delay and capped incremental FPT-time is also true in the parametrised world. Similarly, as in the classical setting [CS17, Prop. 12], the price of a structured delay is the required exponential space of a priority queue.

Theorem 3.4.
For every $a \geq 0$, $\mathsf{CapIncFPT}_{a+1} = \mathsf{IncFPT}_a$.

Proof. '$\supseteq$': Let $E = (Q, \kappa, \mathrm{Sol})$ be a PEP in IncFPT_a via an algorithm $\mathcal{A}$. Let $t\colon \mathbb{N} \to \mathbb{N}$ be a computable function and $p\colon \mathbb{N} \to \mathbb{N}$ be a polynomial as in Definition 3.3 (3".). For every $x \in Q$ algorithm $\mathcal{A}$ outputs i solutions with a running time bounded by

$$\sum_{k=0}^{i} t(\kappa(x)) \cdot p(|x|) \cdot k^a = t(\kappa(x)) \cdot p(|x|) \cdot \sum_{k=0}^{i} k^a \leq t(\kappa(x)) \cdot p(|x|) \cdot (i+1) \cdot i^a \leq 2 \cdot t(\kappa(x)) \cdot p(|x|) \cdot i^{a+1}.$$

Accordingly, we have that $E \in \mathsf{CapIncFPT}_{a+1}$.

'$\subseteq$': Now consider a problem $E = (Q, \kappa, \mathrm{Sol}) \in \mathsf{CapIncFPT}_{a+1}$ via $\mathcal{A}$ enumerating i elements of $\mathrm{Sol}(x)$ in time $< t(\kappa(x))i^{a+1} \cdot p(|x|)$ for all $x \in Q$, for all $0 \leq m \leq |\mathrm{Sol}(x)|$, and some computable function t (see Definition 3.3 (3'.)). We will show that enumerating $\mathrm{Sol}(x)$ can be achieved with an i-th delay of $O(t(\kappa(x)) \cdot p(|x|) \cdot q(i) + s)$ where $q(i) = (i+1)^{a+1} - i^{a+1}$ and s bounds the solution sizes (which is polynomially in the input length; w.l.o.g. let p be an upper bound for this polynomial). To reach this delay, one uses two counters: one for the steps of $\mathcal{A}$ (steps) and one for the solutions initialised with value 1 (solindex). While simulating $\mathcal{A}$, the solutions are inserted into a priority queue Q instead of printing them. Eventually the step counter reaches $t(\kappa(x)) \cdot p(|x|) \cdot \mathsf{solindex}^{a+1}$. Then the first element of Q is extracted, output and solindex is incremented by one. In view of this, $\mathcal{A}$ computed solindex many solutions but only one less has been printed (or $\mathcal{A}$ already halted). Combining these observations leads to calculating the i-th delay:

$$\begin{aligned} & O(t(\kappa(x)) \cdot p(|x|) \cdot (i+1)^{a+1} - t(\kappa(x)) \cdot p(|x|) \cdot i^{a+1} + s) \\ = {} & O(t(\kappa(x)) \cdot p(|x|) \cdot q(i) + p(|x|)) \\ = {} & O(t(\kappa(x)) \cdot p(|x|) \cdot i^a) \qquad (\text{as } q(i) \in O(i^a)) \end{aligned}$$

Clearly, this is a delay of the required form $t(\kappa(x)) \cdot p(|x|) \cdot i^a$, and thereby $E \in \mathsf{IncFPT}_a$. □

Note that from the previous result one can easily obtain the following corollary.

Corollary 3.5.
$\mathsf{CapIncFPT}_1 = \mathsf{DelayFPT}$ *and* $\mathsf{CapIncFPT} = \mathsf{IncFPT}$.

If one drops the restrictions 3. and 4. from Definition 2.7, then Capelli and Strozecki unconditionally show a strict hierarchy for the cap-classes via utilising the well-known time hierarchy theorem [HS65]. Of course, this result transfers also to the parametrised world, that is, to the same generalisation of $\mathsf{CapIncFPT}_a$. Yet it is unknown whether a similar hierarchy can be unconditionally shown for these classes as well as for IncFPT_a. This is a significant question of further research which is strengthened in the following section via connecting parametrised with classical enumeration complexity.

3.2 Connections to Classical Enumeration

Capelli and Strozecki [CS17] ask whether a polynomial delay algorithm using exponential memory can be translated into an output polynomial or even incremental polynomial algorithm requiring only polynomial space. This question might imply a time-space-tradeoff, that is, avoiding exponential space for a DelayP-algorithm will yield the price of an increasing IncP delay. This remark perfectly contrasts with what has been observed by Creignou et al. [Cre+15]. They noticed that outputting solutions ordered by their size seems to require exponential space in case one aims for DelayFPT. As mentioned in the introduction, Meier and Reinbold [MR18] observed how a DelayFPT algorithm with exponential space or a specific problem is transformed into an IncFPT algorithm with polynomial space. These results emphasise why we strive for and why it is valuable to have such a connection between these two enumeration complexity fields. In this section, we will prove that a collapse of IncP and OutputP implies OutputFPT collapsing to IncFPT and *vice versa*.

Capelli and Strozecki [CS17] investigated connections from enumeration complexity to function complexity classes of a specific type. The classes of interest contain many notable computational problems such as integer factoring, local optimisation, or binary linear programming. As we will approach similar classes in the parametrised setting, parametrised variants of these problems are pivotal members of these classes.

It is well known that function variants of classical complexity classes do not contain functions as members but relations instead. Accordingly, we formally identify languages $Q \subseteq \Sigma^*$ and their solution-space $\mathcal{S} \subseteq \Sigma^*$ with relations $\{ (x, y) \mid y \in \mathrm{Sol}(x) \}$ and thereby extend the notation of parametrised problems (pg. 2.2), enumeration problems (Def. 2.7), and parametrised enumeration problems (Def. 2.12). Nevertheless, it is easy to see how to utilise *witness functions* for a given language L such that $x \in L$ implies $f(x) = y$ for some y such that $A(x, y)$ is true, and $f(x) =$"no" otherwise, in order to match the term "function complexity class" more adequately.

Definition 3.6.
We say that a relation $A \subseteq \Sigma^* \times \Sigma^*$ *is* polynomially balanced *if* $(x, y) \in A$ *implies that* $|y| \leq p(|x|)$ *for some polynomial* p.

Observe that, for instances of a (parametrised) enumeration problem E over Σ, the length of its solutions are polynomially bounded. Accordingly, the underlying relation $A \subseteq \Sigma^* \times \Sigma^*$ is already polynomially balanced.

The following two definitions present four function complexity classes.

Definition 3.7 (FP and FNP)**.**
Let $A \subseteq \Sigma^* \times \Sigma^*$ *be a binary and polynomially balanced relation.*

- $A \in \mathsf{FP}$ *if there is a deterministic polynomial time algorithm that, given* $x \in \Sigma^*$*, can find some* $y \in \Sigma^*$ *such that* $A(x, y)$ *is true.*
- $A \in \mathsf{FNP}$ *if there is a deterministic polynomial time algorithm that can determine whether* $A(x, y)$ *is true, given both,* x *and* y.

Definition 3.8 ($\mathsf{F}(\mathsf{FPT})$ and $\mathsf{F}(\mathsf{para\text{-}NP})$)**.**
Let Let $A \subseteq \Sigma^* \times \Sigma^*$ *be a parametrised and polynomially balanced problem with parametrisation* κ.

- $A \in \mathsf{F}(\mathsf{FPT})$ *if there exists a deterministic algorithm that, given* $x \in \Sigma^*$*, can find some* $y \in \Sigma^*$ *such that* $A(x, y)$ *is true and runs in time* $f(\kappa(x)) \cdot p(|x|)$*, where* f *is a computable function and* p *is a polynomial.*
- $A \in \mathsf{F}(\mathsf{para\text{-}NP})$ *if there exists a deterministic algorithm that, given both* x *and* y*, can determine whether* $A(x, y)$ *is true and runs in time* $f(\kappa(x)) \cdot p(|x|)$*, where* f *is a computable function and* p *is a polynomial.*

Note that the definition of $\mathsf{F}(\mathsf{para\text{-}NP})$ follows the verifier characterisation of "precomputation on the parameter" as observed in Corollary 2.6.

The next definition lifts the class $\mathsf{F}(\mathsf{NP} \cap \mathsf{coNP})$ [MP91] to the parameterised setting.

Definition 3.9 ($\mathsf{F}(\mathsf{para\text{-}NP} \cap \mathsf{para\text{-}coNP})$)**.**
Given a language $L \subseteq \Sigma^*$*, we say that* $L \in \mathsf{F}(\mathsf{para\text{-}NP} \cap \mathsf{para\text{-}coNP})$ *if there exist two parametrised and polynomially balanced problems* $A \subseteq \Sigma^* \times \Sigma^*$ *and* $B \subseteq \Sigma^* \times \Sigma^*$ *with parametrisations* κ *and* κ' *as well as a nondeterministic algorithm* N *such that the following two properties are fulfilled.*

- *For each* $x \in L$ *either there exists a* y *with* $(x, ay) \in A$*, or there is a* z *with* $(x, bz) \in B$*, where* $a \neq b$ *are two special markers in* Σ.
- *Given* $x \in \Sigma^*$*,* N *either can find a* y *with* $A(x, ay)$ *or a* z *with* $B(x, bz)$ *in time* $f(\kappa(x)) \cdot p(|x|) + g(\kappa'(x)) \cdot q(|x|)$*, or state that such ones doe not exist, where* p, q *are polynomials and* f, g *are computable functions.*

The typical problems then ask for finding an appropriate witness with respect to a given x.

Class	machine	runtime	constraints
FP	det.	$p(\|x\|)$	find y s.t. $A(x, y)$
FNP	nond.	$p(\|x\|)$	guess y s.t. $A(x, y)$
TF(NP)	nond.	$p(\|x\|)$	guess y s.t. $A(x, y)$, A is total
F(FPT)	det.	$f(\kappa(x)) \cdot p(\|x\|)$	κ parametrisation, find y s.t. $A(x, y)$
F(para-NP)	nond.	$f(\kappa(x)) \cdot p(\|x\|)$	κ parametrisation, guess y s.t. $A(x, y)$
TF(para-NP)	nond.	$f(\kappa(x)) \cdot p(\|x\|)$	κ parametrisation, guess y s.t. $A(x, y)$, A is total
F(para-NP $\cap$ para-coNP)	nond.	$f(\kappa(x)) \cdot p(\|x\|) + g(\kappa(x)') \cdot q(\|x\|)$	relations A, B with parametrisations κ and κ', either find y with $A(x, ay)$ or z with $B(x, bz)$

Tab. 3.1: Overview of function complexity classes. In the machine column 'det.'/'nond.' abbreviates 'deterministic'/'nondeterministic'. In the runtime column p and q are polynomials, f and g are two computable functions, κ is the parameter, and x is the input.

In 1994, Bellare and Goldwasser [BG94] investigated functional versions of NP problems. They observed that under standard complexity-theoretic assumptions (deterministic doubly exponential time is different from nondeterministic doubly exponential time) these problems are not self-reducible. Bellare and Goldwasser noticed that these functional versions are harder than their corresponding decision variants.

A binary relation $R \subseteq \Sigma^* \times \Sigma^*$ is said to be *total* if for every $x \in \Sigma^*$ there exists a $y \in \Sigma^*$ such that $(x, y) \in R$.

Definition 3.10 (Total function complexity classes)**.**
The class TF(NP)*, resp.,* TF(para-NP)*, is the restriction of* FNP*, resp.,* F(para-NP)*, to total relations.*

The two previously defined classes are *promise classes* in the sense that the existence of a witness y with $A(x, y)$ is guaranteed. Furthermore, defining a class TF(P) or TF(FPT) is not meaningful as it is known that FP $\subseteq$ TF(NP) (see, e.g., the work of Johnson et al. [JPY88a, Lemma 3] showing that FP is contained in PLS, polynomial local search, which is contained in TF(NP) by Megiddo and Papdimitriou [MP91, p. 319]). Similar arguments apply to F(FPT) $\subseteq$ TF(para-NP)).

Now, we can define a generic (parametrised) search and a generic (parametrised) enumeration problem. Note that the parameter is only relevant for the parametrised counterpart named in brackets.

Problem:	(para-)ANOTHERSOL$_A$, where $A \subseteq \Sigma^* \times \Sigma^*$
Input:	$x \in \Sigma^*$, $S \subseteq \Sigma^*$.
Parameter:	$\kappa\colon \Sigma^* \to \mathbb{N}$.
Task:	output y in $\mathrm{Sol}(x) \setminus S$, or answer $S \supseteq \mathrm{Sol}(x)$.

Problem:	(para-)ENUM-A, where $A \subseteq \Sigma^* \times \Sigma^*$
Input:	$x \in \Sigma^*$.
Parameter:	$\kappa\colon \Sigma^* \to \mathbb{N}$.
Output:	output all y with $A(x, y)$.

The two results of Capelli and Strozecki [CS17, Prop. 7 and 9] which are crucial in the course of this section are restated in the following.

Proposition 3.11 (Prop. 7 in [CS17])**.**
Let $A \subseteq \Sigma^* \times \Sigma^*$ be a binary relation such that, given $x \in \Sigma^*$, one can find a y with $A(x, y)$ in deterministic polynomial time. ANOTHERSOL$_A \in$ FP *if and only if* ENUM-$A \in$ CapIncP.

Proposition 3.12 (Prop. 9 in [CS17])**.**
TF(NP) = FP *if and only if* OutputP = CapIncP.

In 1991, Megiddo and Papadimitriou studied the complexity class TF(NP) [MP91]. In a recent investigation, Goldberg and Papadimitriou introduced a rich theory around this complexity class that features also several aspects of proof theory [GP17]. Megiddo and Papadimitriou prove the following result for the classes F(NP ∩ coNP) and TF(NP). It is easily lifted to the parametrised setting.

Theorem 3.13.
F(para-NP ∩ para-coNP) = TF(para-NP).

Proof. We restate the classical proofs of Megiddo and Papadimitriou [MP91].

"⊆": By definition of the class F(para-NP ∩ para-coNP), either there exists a y with $(x, ay) \in A$, or there is a z with $(x, bz) \in B$. As a result, the mapping $(A, B) \mapsto A \cup B$ suffices and $A \cup B$ is total. So we just need to guess which of A or B to choose.

"⊇": the mapping $A \mapsto (A, \emptyset)$ is obvious. □

For the subsequently proven lemma (which is the parametrised pendant of Prop. 3.11) and theorem we follow the argumentation in the proofs of the classical results (Prop. 3.11 and 3.12).

Lemma 3.14.
Let $A \subseteq \Sigma^*$ be a param. problem with parametrisation κ. Then, para-ANOTHERSOL$_A \in$ F(FPT) *if and only if* para-ENUM-$A \in$ CapIncFPT.

Proof. "⇒": Let para-ANOTHERSOL$_A \in$ F(FPT) via some algorithm $\mathcal{A}$. Algorithm 3.1 shows that para-ENUM-$A \in$ CapIncFPT. The runtime of each step is $f(\kappa(x)) \cdot p(|x|, |S|)$

Algorithm 3.1: Algorithm showing para-ENUM-$\mathcal{A} \in$ CapIncFPT.

```
1 S ← ∅;
2 repeat
3 |   y ← A(x, S);
4 |   S ← S ∪ {y};
5 |   print y;
6 until S = A(x);
```

for some polynomial p and some computable function f. Consequently, this shows that para-ENUM-$\mathcal{A} \in$ CapIncFPT.

"$\Leftarrow$": Let para-ENUM-$\mathcal{A} \in$ CapIncFPT. Then, there exists a parametrised enumeration algorithm $\mathcal{A}$ that, given input $x \in \Sigma^*$, outputs $i \leq \text{Sol}(x)$ elements in a runtime of $f(\kappa(x)) \cdot i^a \cdot p(|x|)$ for some computable function f, $a \in \mathbb{N}$, and polynomial p.

Now, we explain how to compute para-ANOTHERSOL$_{\mathcal{A}}$ in fpt-time. Given (x, S), simulate $\mathcal{A}$ for $f(\kappa(x)) \cdot (|S|+1)^a \cdot p(|x|)$ steps. If the simulation successfully halts then $\text{Sol}(x)$ is completely output. Just search a $y \in \text{Sol}(x) \setminus S$ or output "$S \supseteq \text{Sol}(x)$". Otherwise, if $\mathcal{A}$ did not halt then it did output at least $|S|+1$ different elements. Finally, we just compute $A(x) \setminus S$ and print a new element. □

The next theorem translates the result of Proposition 3.12 from classical enumeration complexity to the parametrised setting.

Theorem 3.15.
TF(para-NP) = F(FPT) *if and only if* OutputFPT = CapIncFPT.

Proof. "$\Leftarrow$": Let $A(x, y) \in$ TF(para-NP) be a parametrised language over $\Sigma^* \times \Sigma^*$ with parametrisation κ and M be the corresponding nondeterministic algorithm running in time $g(\kappa(x)) \cdot p(|x|)$ for a polynomial p, a computable function g, and input x. Now, define the relation $C \subseteq \Sigma^* \times \left\{ y\#w \mid y \in \Sigma^*, w \in \{0,1\}^* \right\}$ such that

$$C(x, y\#w) \text{ if and only if } A(x, y) \text{ and } |w| \leq p(|x|).$$

Then for each x there exists $y\#w$ such that $C(x, y\#w)$ is true by definition of TF(para-NP). Via padding, for each y, we have $|C(x, y\#w)| \geq 2^{p(|x|)}$ such that $A(x, y)$ is true. By construction, the trivial brute-force enumeration algorithm checking all $y\#w$ is in fpt-time for every element of $\text{Sol}(x)$. Accordingly, this gives para-ENUM-C $\in$ OutputFPT as the runtime for OutputFPT algorithms encompasses $|\text{Sol}(x)|$ as a factor.

Then para-ENUM-C $\in$ CapIncFPT and the first $y\#w$ is output in fpt-time. Since A was arbitrary, we conclude with TF(para-NP) = F(FPT) (as F(FPT) $\subseteq$ TF(para-NP) contains only total relations).

"$\Rightarrow$": Consider a problem para-ENUM-$\mathcal{A} \in$ OutputFPT with ENUM-$\mathcal{A} = (Q, \kappa, \text{Sol})$ and $A \subseteq \{0,1\}^* \times \{0,1\}^*$. For every $x \in Q$ and $S \subseteq \text{Sol}(x)$ let $D((x, S), y)$ be true if and only if either ($y \in \text{Sol}(x) \setminus S$) or ($y = \#$ and $S \supseteq \text{Sol}(x)$). Then $D \in$ TF(para-NP):

Language	Class	Machine	Runtime
B	TF(para-NP)	M	nondet. $f(\kappa(x)) \cdot p(\lvert x\rvert)$
D_B	para-NP	$M(D_B)$	nondet. $g(\kappa(x)) \cdot q(\lvert x\rvert)$
D_B^Q	NP	$M(D_B^Q)$	nondet. $r(\lvert\pi(\kappa(x))\rvert, \lvert x\rvert)$

Tab. 3.2: Machines in the direction '$\Leftarrow$' in the proof of Theorem 3.16. In the runtime column, p, q, r are polynomials, f, g are computable functions.

(1.) D is total by construction,

(2.) as para-ENUM-$\mathcal{A}$ is a parametrised enumeration problem, there exists a polynomial q such that for every solution $y \in \mathrm{Sol}(x)$ we have $|y| \leq q(|x|)$, and

(3.) finally, we need to show that $D((x, S), y)$ can be verified in deterministic time $f(\kappa(x)) \cdot p(|x|, |S|, |y|)$ for a computable function f and a polynomial p.

Case $y \neq \#$: $D((x, S), y)$ is true if and only if $y \in \mathrm{Sol}(x) \setminus S$. This requires testing whether $y \notin S$ and $y \in \mathrm{Sol}(x)$. Both can be tested in polynomial time: $p(|y|, |S|)$, respectively, $p(|x|)$ which follows from Def. 2.12 (4.).

Case $y = \#$: $D((x, S), y)$ is true if and only if $S \supseteq \mathrm{Sol}(x)$. As para-ENUM-$\mathcal{A} \in$ OutputFPT there is an algorithm $\mathcal{A}$ outputting $\mathrm{Sol}(x)$ in $f(\kappa(x)) \cdot p(|x|, |\mathrm{Sol}(x)|)$ steps. Now, run $\mathcal{A}$ for at most $f(\kappa(x)) \cdot p(|x|, |S|)$ steps. Then finishing timely implies that $\mathrm{Sol}(x)$ is completely generated and we merely check $S \supseteq \mathrm{Sol}(x)$ in time polynomial in $|S|$. If $\mathcal{A}$ did not halt during its simulation we can deduce $|\mathrm{Sol}(x)| > |S|$. Accordingly, $S \not\supseteq \mathrm{Sol}(x)$ follows and $D((x, S), y)$ is not true.

As TF(para-NP) = FPT, given a (x, S), we can compute a y with $y \in \mathrm{Sol}(x) \setminus S$ or decide there is none (and then set $y = \#$) in fpt-time. Accordingly, para-ANOTHERSOL$_{\mathcal{A}}$ is in F(FPT) and, by applying Lemma 3.14, we get para-ENUM-$\mathcal{A}$ is in CapIncFPT. This settles that OutputFPT = CapIncFPT and concludes the proof. □

The next theorem builds on previous statements in order to connect a collapse in the classical function world to a collapse in the parametrised function world.

Theorem 3.16.
TF(para-NP) = F(FPT) *if and only if* TF(NP) = FP.

Proof. "$\Leftarrow$": If TF(NP) = FP then every total relation $A \in$ TF(NP) over $\Sigma^* \times \Sigma^*$ has a deterministic polynomial time algorithm that, given $x \in \Sigma^*$, can find some $y \in \Sigma^*$ such that $A(x, y)$ is true. Now choose some $B \in$ TF(para-NP) = F(para-NP$\cap$para-coNP) (Theorem 3.13) via machine M running in time $f(\kappa(x)) \cdot p(|x|)$ for polynomial p, computable function f, parametrisation κ, and input x. The corresponding parametrised decision variant D_B is in para-NP via the machine $M(D_B)$ which simulates M and accepts or rejects depending on having a witness ay or bz. In particular, by Proposition 2.5, we know that there exists a computable function $\pi\colon \mathbb{N} \to \Sigma^*$ and a problem $D_B^Q \subseteq \Sigma^* \times \Sigma^*$ such that $D_B^Q \in$ NP and the

following is true: for all instances $x \in \Sigma^*$ we have that $x \in D_B$ if and only if $(x, \pi(\kappa(x))) \in D_B^Q$. Let $M(D_B^Q)$ be the machine that shows D_B^Q is in NP, and as B is total, the unparameterised version B' of B is in $\mathsf{TF}(\mathsf{NP})$ via the same machine M. By assumption, B' is also in FP and then M runs in deterministic polynomial time. Accordingly, we define a machine $\widetilde{M}$ for B which, given input $x \in \Sigma^*$, computes $\pi(\kappa(x))$, then simulates $M(D_B^Q)$ on $(x, \pi(\kappa(x)))$, and runs in time

$$f_\pi(\kappa(x)) + r(|\pi(\kappa(x)|, |x|) \qquad (\star)$$

where $f_\pi \colon \Sigma^* \to \mathbb{N}$ is a computable function that estimates the runtime of computing $\pi(\kappa(x))$. Equation $(\star)$ clearly is an fpt-runtime showing that we have $B \in \mathsf{F}(\mathsf{FPT})$. Accordingly, we can deduce that $\mathsf{TF}(\mathsf{para\text{-}NP}) = \mathsf{F}(\mathsf{FPT})$ as B was chosen arbitrarily.

"$\Rightarrow$": If $\mathsf{TF}(\mathsf{para\text{-}NP}) = \mathsf{F}(\mathsf{FPT})$ then every total relation $A \in \mathsf{TF}(\mathsf{para\text{-}NP})$ over $\Sigma^* \times \Sigma^*$ has a deterministic algorithm that, given $x \in \Sigma^*$, can find some $y \in \Sigma^*$ such that $A(x, y)$ is true, and which runs in time $f(\kappa(x)) \cdot p(|x|)$ for a computable function f, a polynomial p, and the parametrisation κ of A. Now choose some $B \in \mathsf{TF}(\mathsf{NP}) = \mathsf{F}(\mathsf{NP} \cap \mathsf{coNP})$ [MP91]. The corresponding decision variant D_B is in NP via the machine M which accepts or rejects depending on having a witness ay or bz. In particular, for every language $L \in \mathsf{NP}$, L belongs to para-NP for the trivial parametrisation $\kappa(x) = 1$ for all $x \in \Sigma^*$. It follows that D_B, parametrised in that specific way, is in FPT with a runtime of $f(1) \cdot q(|x|)$ where q is a polynomial and f a computable function, so $D_B \in \mathsf{P}$. Consequently, the machine M from above then actually runs in polynomial time and even computes the function variant B. Accordingly, we can deduce that $\mathsf{TF}(\mathsf{NP}) = \mathsf{FP}$ as the language B was chosen arbitrarily. □

Combining Theorem 3.15 with Theorem 3.16 and finally Proposition 3.12, connects the parametrised enumeration world with the classical enumeration world.

Corollary 3.17.
OutputFPT = CapIncFPT *if and only if* OutputP = CapIncP.

If one does not consider space requirements, we can deduce the following corollary by applying Corollary 3.5 and Proposition 2.10.

Corollary 3.18.
OutputFPT = IncFPT *if and only if* OutputP = IncP.

Now, the observations made by Capelli and Strozecki [CS17] have directly been transferred to our setting. Accordingly, for instance, the existence of one-way functions would separate OutputFPT from IncFPT as well. Also a collapse of OutputFPT to CapIncFPT would yield a collapse of $\mathsf{TF}(\mathsf{NP})$ to FP (Prop. 3.12) and as well as of P to $\mathsf{NP} \cap \mathsf{coNP}$ (due to $\mathsf{TF}(\mathsf{NP}) = \mathsf{F}(\mathsf{NP} \cap \mathsf{coNP})$ [MP91]).

3.3 CardinalitySAT

In the last section of this chapter, we want to close with some observations on a variant of SAT where the size of the solution space (that is, all satisfying assignments of a given formula) is the parameter. The results from this section a rather negative as they will clearly point out why this problem is not of much help in separating any of the previously considered classes. Now, we will define the parametrised enumeration version of the mentioned SAT variant.

Problem:	ENUM-CARDINALITYSAT
Input:	Propositional formula φ, $k \in \mathbb{N}$.
Parameter:	The integer k.
Output:	All sets of assignments Θ such that $\lvert\Theta\rvert = k$ and $\theta \models \varphi$ for each $\theta \in \Theta$.

Observe that we ask for an enumeration of all satisfying assignments if the input is $(\varphi, 1)$. At the other extremum, namely if the input is $(\varphi, 2^{\mathrm{Vars}(\varphi)})$, then we ask to output the unique single solution containing all assignments if φ is tautological. As a result, the corresponding decision problem is NP- and coNP-hard.

Definition 3.19 ([BG82])**.**
The class US *(unique solution) is the class of all languages* $L \subseteq \Sigma^*$ *which can be represented as*

$$L = \{ x \in \Sigma^* \mid \textit{there exists exactly one } y \in \Delta^* \textit{ s.t. } R(x, y) \},$$

for some $R \in \mathsf{FP}$ *and some solution alphabet* Δ*.*

A natural complete problem for US is UNIQUESAT, the set of all propositional formulas which have exactly one satisfying assignment. As US is closed under complementation and UNIQUESAT is also coNP-hard it follows $\mathsf{coNP} \subseteq \mathsf{US} \subseteq \mathsf{DP}$ (the class of differences of NP-languages). It is unknown if $\mathsf{NP} \overset{?}{\subseteq} \mathsf{US}$ or $\mathsf{US} \overset{?}{=} \mathsf{DP}$ is true. Subsequently we will show that is unlikely that ENUM-CARDINALITYSAT is in IncFPT.

Theorem 3.20.
ENUM-CARDINALITYSAT $\in$ IncFPT *implies*
(1.) $\mathsf{P} = \mathsf{NP}$*, and (2.)* $\mathsf{US} = \mathsf{P}$.

Proof. (1.) If ENUM-CARDINALITYSAT $\in$ IncFPT then the instance $(\varphi, 1)$ is solved with the i-th delay of $t(1) \cdot p(|\varphi|, i)$. That being the case, the first solution is output in polynomial time.

(2.) If ENUM-CARDINALITYSAT $\in$ IncFPT then every formula φ having a unique satisfying assignment can be tested for satisfiability in time $t(1) \cdot p(|\varphi|, 1)$.

□

Similarly we can see that a membership of ENUM-CARDINALITYSAT in the larger class OutputFPT is rather unlikely as well. The result is proven similarly to the previous theorem.

Corollary 3.21.
ENUM-CARDINALITYSAT $\in$ OutputFPT *implies* US $=$ P.

Page left intentionally blank to have matching page numbers with the printed version.

Principles of Parametrised Enumeration

4

> *There is no coherent knowledge, i.e. no uniform comprehensive account of the world and the events in it. There is no comprehensive truth that goes beyond an enumeration of details, but there are many pieces of information, obtained in different ways from different sources and collected for the benefit of the curious. The best way of presenting such knowledge is the list — and the oldest scientific works were indeed lists of facts, parts, coincidences, problems in several specialised domains.*
>
> **— Paul Karl Feyerabend**

Parametrised Enumeration can be seen as joining two significant areas of research: *Parametrised Complexity Theory* together with *Enumeration Algorithms*. In this chapter, we will focus on two significant paradigms where each single one stems from either of these areas. From the first, we will investigate the technique of *kernelisation* [Cyg+15, Cha. 2 and 9]. This method is utilised to design parameter-efficient algorithms and can even characterise the class FPT: Having a kernelisation allows for computing and brute-forcing it; otherwise, having an FPT algorithm enables us to either output a trivial kernel after solving the instance directly, or the instance is already a kernel. One will see how this proof idea reassembles in showing the correctness of Theorem 4.3.

The other mentioned technique is an algorithmic paradigm known as *self-reducibility* [Sch76; KV91; Sch09]. Many enumeration algorithms base on this type of strategy and, in essence, corresponding problems allow for a "search-reduces-to-decision" algorithm to obtain its solutions. In particular, satisfiability related problems often enables one for this kind of algorithms. We have already seen a basic example for this principle in Example 2.11.

In the following sections, we will examine how both principles can be utilised in the setting of *Parametrised Enumeration*.

4.1 Kernelisation

Structurally, the task of kernelisation consists of a pre-processing step. Formally, this step can be seen as a polynomial time many-one reduction of a problem to

itself with an additional property: the (size of the) image is bounded in terms of the parameter of the argument (see [FG06]). More formally, in the usual sense a kernelisation K of a problem Q is a mapping which preserves membership in Q and bounds the size of the kernel in some way (which is the following Def. 4.1 (1.)). In our setting we extend this notion to an *enum-kernelization*. This should then be seen as a pre-processing step suitable for efficient enumeration.

Definition 4.1.
Let $(Q, \kappa, \mathrm{Sol})$ *be a parameterized enumeration problem over* Σ. *A polynomial time computable function* $K\colon \Sigma^* \to \Sigma^*$ *is an* enum-kernelization *of* $(Q, \kappa, \mathrm{Sol})$ *if there exist:*

(1.) a computable function $h\colon \mathbb{N} \to \mathbb{N}$ *such that for all* $x \in \Sigma^*$ *we have*

$$(x \in Q \Leftrightarrow K(x) \in Q) \text{ and } |K(x)| \leq h(\kappa(x)),$$

(2.) a computable function $f\colon \Sigma^* \times \Sigma^* \to \mathcal{P}(\Sigma^*)$ *which, given a pair* (x, w) *where* $x \in Q$ *and* $w \in \mathrm{Sol}(K(x))$, *computes a subset of* $\mathrm{Sol}(x)$, *such that*

a) for all $w_1, w_2 \in \mathrm{Sol}(K(x))$, $w_1 \neq w_2 \Rightarrow f(x, w_1) \cap f(x, w_2) = \emptyset$,

b) $\biguplus_{w \in \mathrm{Sol}(K(x))} f(x, w) = \mathrm{Sol}(x)$, $f(x, w) = \emptyset$ *if* $w \notin \mathrm{Sol}(K(x))$, *and*

c) there exists an enumeration algorithm $\mathcal{A}_f$, *which on input* (x, w), *where* $x \in Q$ *and* $w \in \mathrm{Sol}(K(x))$, *outputs all solutions of* $f(x, w)$ *with delay* $p(|x|) \cdot t(\kappa(x))$, *where* p *is a polynomial and* t *is a computable function.*

If K *is an enum-kernelization of* $(Q, \kappa, \mathrm{Sol})$, *then for every instance* x *of* Q *the image* $K(x)$ *is called an* enum-kernel *of* x *(under* K*).*

Visually, an enum-kernelization can be seen as a reduction K from a parameterised enumeration problem to itself. Similar to the decision setting it has the property that the image is bounded in terms of the parameter argument (Def. 4.1 (1.)). For a problem instance x, $K(x)$ is the kernel of x. Observe that if K is an enum-kernelisation of the enumeration problem $(Q, \kappa, \mathrm{Sol})$, then it is also a kernelisation for the associated decision problem. Additionally, enum-kernelisations have an additional property: the set of solutions of the original instance x can be rebuilt from the set of solutions of the image $K(x)$ with DelayFPT (Def. 4.1 (2.) c)). This generalises the notion of *full kernels* [Dam06; FSV13]) appearing in the context of what is called *subset minimization problems*. A full kernel is a kernel that contains all minimal solutions since they represent in a certain way all solutions. In Subsection 4.2.2, in the context of backdoor sets, a *loss-free kernel* [SS08] is a similar notion. In our definition, an enum-kernel is a kernel that represents all solutions in the sense that they can be obtained with FPT delay from the solutions for the kernel. This observation is visualised in Figure 4.1.

Given an undirected graph G and a positive integer k. Vertex-Cover then asks whether there exists $\leq k$ different vertices in G such that all edges are "covered" (for each edge at least one endpoint is chosen). In fact, Vertex-Cover is one of

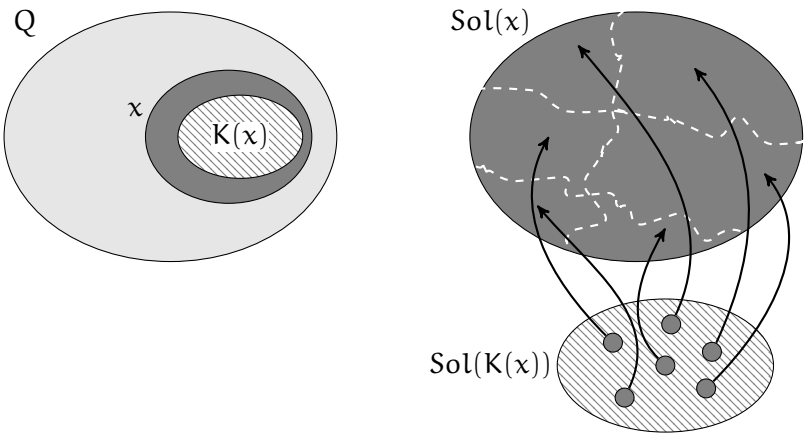


Fig. 4.1: Enum-kernelisation example. The grey circles in $\mathrm{Sol}(K(x))$ indicate the different solutions which partition the solution set $\mathrm{Sol}(x)$

the most investigated problems in *Parametrised Complexity* [Ang+16]. Specifically, for kernelisation it is the standard example. Further on, we examine it in the light of our definition of enum-kernelisation. Formally, the parametrised enumeration version of the problem is defined as follows.

Problem:	ENUM-VERTEX-COVER
Input:	An undirected graph G and a positive integer k.
Parameter:	The integer k.
Output:	The set of all vertex covers of G of size $\leq k$.

Theorem 4.2.
ENUM-VERTEX-COVER *has an enum-kernelization.*

Proof. Given a graph $G = (V, E)$ and a positive integer k, we are interested in enumerating all vertex covers of G of size at most k. We prove that the famous Buss' kernelisation[1] provides an enum-kernelisation. Let us recall that Buss' algorithm consists in applying repeatedly the following rules until no more reduction can be made:

(1.) If v is a vertex of degree greater than k, remove v from the graph and decrease k by one.

(2.) If v is an isolated vertex, remove it.

As a consequence, in the process the value of k eventually decreases. Let us denote with $k' \leq k$ the value which is reached after the algorithm terminates. If the rule (1.) is applied ℓ times, then $k' = k - \ell$. Then the kernel $K(G, k)$ is the reduced graph (V_K, E_K) so obtained if it has less than $(k')^2$ edges, and the complete graph $\mathcal{K}_{k'+2}$ otherwise. Consequently, the second part of Def. 4.1 (1.) is fulfilled as the size of the kernel is bound by some function in k, that is $h(k) = (k+2)^2$.

[1] Jonathan F. Buss and Judy Goldsmith acknowledge Samuel R. Buss for this algorithm via a personal communication reference [BG93].

Now we show that the first part of Def. 4.1 (1.) is fulfilled. Whenever in a certain step rule (1.) is applicable to a vertex v, and v is not removed immediately, then rule (1.) remains applicable to v also in any further step. This is valid until it is eventually removed. This is correct because removing another vertex v' by application of rule (1.) instead of v decreases the value of k by one and the degree of v by at most one. Accordingly, whenever we have a choice during the removal process, our choice does not influence the finally obtained graph: the kernel is unique.

Now we turn towards the function f required from Def. 4.1 (2.) which produces the solutions for the original instance. On that account suppose that $K(G,k) = (V_K, E_K)$ is the computed kernel and k' is its parameter value. Let V_D be the set of vertices (of large degree) that are removed from G by rule (1.), and V_I be the set of (isolated) vertices that are removed from G by rule (2.). On the one hand, every vertex cover of size $\leq k$ of G has to contain V_D. On the other hand, no vertex from V_I is part of a minimal vertex cover as no edges are incident. As a consequence, all vertex covers W of G are obtained by considering all the vertex covers of $K(G,k)$ of size $\leq k' = k - |V_D|$, completing them by V_D and by some vertices of V_I up to the cardinality k. That being so we have to consider all subsets of V_I of size bounded by $k - (|W| + |V_D|)$. Given a vertex cover W of $K(G,k)$ we define

$$f(G,W) = \{ W \uplus V_D \uplus V' \mid V' \subseteq V_I, |V'| \leq k - |W| - |V_D| \}.$$

Let $W_1 \neq W_2$ be two given solution vertex covers such that $W_1, W_2 \in \mathrm{Sol}(K(G,k))$. Clearly, a) of Def. 4.1 (2.) is fulfilled, that is, $f(G,W_1) \cap f(G,W_2) = \emptyset$, as $W_1 \neq W_2$ implies $W_1 \uplus V_D \uplus V' \neq W_2 \uplus V_D \uplus V'$ for V' and V_D as above. Also b) is true as

$$\bigcup_{W \in \mathrm{Sol}(K(G,k))} f(G,W) = \mathrm{Sol}(G,k)$$

is the set of all $\leq k$-vertex covers of G that emerges from the unique kernel $K(G,k)$. The reason for that is that each $W \in \mathrm{Sol}(K(G,k))$ is associated with a different combination of the vertices in $V' \subseteq V_I$. Finally, with respect to c), given a vertex cover W of $K(G,k)$, we explain the behaviour of the enumeration algorithm. At first we have a polynomial time pre-processing of G by Buss' kernelization in order to compute V_D and V_I. Then, the enumeration of $f(G,W)$ comes down to an enumeration of all subsets of V_I of size at most $k' = k - |W| - |V_D|$. Enumerating all subsets of V_I of size at most k' can be done with delay $O(k')$ (and consequently with FPT-delay) by a standard recursive algorithm outputting the subsets in lexicographic order. □

As in the context of decision problems, enum-kernelisation actually characterises the class of enumeration problems having a DelayFPT-algorithm, as shown in the following theorem.

Theorem 4.3.
For every parameterized enumeration problem $(Q, \kappa, \mathrm{Sol})$ *over* Σ*, the following are equivalent:*

(1.) $(Q, \kappa, \mathrm{Sol})$ *has an enum-kernelisation.*

(2.) $(Q, \kappa, \mathrm{Sol})$ *is in* DelayFPT

Proof. "(1.) $\Rightarrow$ (2.)": Let K be an enum-kernelisation of $(Q, \kappa, \mathrm{Sol})$ computable in time p' for some polynomial p', and Sol be computable in time g for some computable function g. W.l.o.g. assume that g is non-decreasing. Furthermore, let h, f, t be computable functions, $\mathcal{A}_f$ be an enumeration algorithm, and p be a polynomial, all given as in Definition 4.1.

Starting with an instance $x \in \Sigma^*$ the following algorithm enumerates all solutions in $\mathrm{Sol}(x)$ with DelayFPT:

(1.) Compute $K(x)$ in polynomial time $p'(|x|)$.

(2.) Compute $\mathrm{Sol}(K(x))$ in time $g(h(\kappa(x)))$ since $|K(x)| \leq h(\kappa(x))$.

(3.) For each $w \in \mathrm{Sol}(K(x))$ use $\mathcal{A}_f$ on the input (x, w) with delay $p(|x|) \cdot t(\kappa(x))$.

The delay of this enumeration algorithm is bounded from above by $p'(|x|) + g(h(\kappa(x))) + p(|x|) \cdot t(\kappa(x)$, which is bounded from above by $(p'(|x|) + p(|x|)) \cdot (g(\kappa(x)) + t(\kappa(x)))$, and consequently proves that this is a DelayFPT algorithm. The correctness of the algorithm follows from the Definition 4.1 of an enum-kernelisation: Item (2.) a) ensures that there is no repetition, Item (2.) b) that all solutions are output.

"(2.) $\Rightarrow$ (1.)": Let $\mathcal{A}$ be an enumeration algorithm for $(Q, \kappa, \mathrm{Sol})$ that requires delay $p(n) \cdot t(k)$ where p is a polynomial and t is some computable function. W.l.o.g. assume that $p(n) \geq n$ for all positive integers n. If $Q = \emptyset$ or $Q = \Sigma^*$ then $(Q, \kappa, \mathrm{Sol})$ has a trivial kernelisation that maps every $x \in \Sigma^*$ to the empty string ϵ.

Case "$Q = \emptyset$": ✓

Case "$Q = \Sigma^*$": fix $w_\epsilon \in \mathrm{Sol}(\epsilon)$ and set for all x, $f(x, w_\epsilon) = \mathrm{Sol}(x)$ and $f(x, w) = \emptyset$ for $w \in \mathrm{Sol}(\epsilon) \setminus \{w_\epsilon\}$.

Now consider the interesting case, that is, $\emptyset \subsetneq Q \subsetneq \Sigma^*$. In that case fix $x_0 \in \Sigma^* \setminus Q$, and $x_1 \in Q$ with $w_1 \in \mathrm{Sol}(x_1)$. The following algorithm $\mathcal{A}'$ then computes an enum-kernelisation for $(Q, \kappa, \mathrm{Sol})$ and given $x \in \Sigma^*$ with $n = |x|$ and $k = \kappa(x)$:

(1.) Simulate $p(n) \cdot p(n)$ steps of $\mathcal{A}$.

(2.) If it stops with the answer "no solution", then set $K(x) = x_0$ (since $x_0 \notin Q$, the function f does not need to be defined).

(3.) If a solution is output within this time, then set $K(x) = x_1$, $f(x, w_1) = \mathrm{Sol}(x)$ and $f(x, w) = \emptyset$ for all $w \in \mathrm{Sol}(x_1) \setminus \{w_1\}$.

(4.) If it does not output a solution within this time, then $n \leq p(n) \leq t(k)$ (because of $\mathcal{A}$'s delay of $p(n) \cdot t(k)$) and then we set $K(x) = x$, and $f(x, w) = \{w\}$ for all $w \in \mathrm{Sol}(x)$.

The kernel $K(x)$ can consequently be computed in time $p(n)^2$, $|K(x)| \leq |x_0| + |x_1| + t(k)$. The membership in Q is preserved: $(x \in Q \Leftrightarrow K(x) \in Q)$. Finally, the function f we have obtained satisfies all the requirements of Definition 4.1. In particular, the computation of $f(x, w)$ in FPT (1.–4.) completely describes the behaviour of $\mathcal{A}_f$. As a result, K provides indeed an enum-kernelisation for (Q, κ, Sol). □

Corollary 4.4.
ENUM-VERTEX-COVER *is in* DelayFPT.

Observe that in the proof of Theorem 4.2, the enumeration of the sets of solutions obtained from a solution W of $K(G)$ is enumerable even with polynomial-delay, we do not need FPT-delay as already has been witnessed by Theorem 3.1.

4.2 Self-Reducibility and Bounded-Search-Trees

Let us recall that satisfiability related problems often allow for applying the technique of self-reducibility. In view of this, we will first investigate the enumeration of models of a propositional formula having weight at least k. Afterwards we turn to the concept of backdoor sets. Informally, with the parameter backdoor size one tries to overcome a small distance of a formula being tractable. This definition was first introduced by Golmes, Williams and Selman [WGS03] to model short distances to efficient subclasses. Until today backdoors gained copious attention in many different areas: abduction [PRS13], answer set programming [FS15b; FS15a], argumentation [DOS12], default logic [FMS16], temporal logic [Mei+16], planning [KOP15], and constraint satisfaction [Gas+17].

In our setting, we will investigate enumeration of strong HORN-backdoor sets of size k. In the first example, the underlying decision problem can be solved by using kernelisation (see Kratsch et al. [KMW16]), while in the second it is solved by using the bounded-search-tree technique.

4.2.1 Enumeration Complexity of Max-Ones-SAT

Previously, self-reducibility has been successfully used to enumerate all satisfying assignments of a generalised CNF-formula [CH97]. Creignou and Hébrard identified subclasses of general CNF-formulas which admit efficient enumeration algorithms. In the parametrised complexity setting a natural problem to study is MAXONES-SAT. Here, ones has to take the weight of an assignment into account, that is, the number of variables set to true. In particular, one aims for assignments which have weight at least k while k is the underlying parameter. Now we want to study the corresponding enumeration problem and will investigate it with respect to Schaefer's framework [Sch78]. Before we can state the problem formally we need to introduce some further notions.

Definition 4.5 (Relations and constraints)**.**
A logical relation *of arity* k *is a relation* $R \subseteq \{0,1\}^k$. *A* constraint*,* C*, is a formula*

$C = R(x_1, \ldots, x_k)$*, where* R *is a logical relation of arity* k *and the* x_i*'s are (not necessarily distinct) variables. If* u *and* v *are two variables, then* $C[u/v]$ *denotes the constraint obtained from* C *in replacing each occurrence of* v *by* u*. If* V *is a set of variables, then* $C[u/V]$ *denotes the set of constraints where* u *is substituted to every occurrence of every variable of* V *in* C*.*

Abusing notation we do not differentiate between a relation and its predicate symbol.

Definition 4.6 (Constraint language)**.**
Let V *be a set of variables. An assignment* $m\colon V \to \{0,1\}$ satisfies *a constraint* C *if* $(m(x_1), \ldots, m(x_k)) \in R$*. A* constraint language Γ *is a finite set of logical relations. A* Γ-formula ϕ*, is a conjunction of constraints using only logical relations from* Γ*. As a consequence, it is a quantifier-free first order formula. With* $\mathbf{Vars}(\phi)$ *we denote the set of variables appearing in* ϕ*. A* Γ*-formula* ϕ *is satisfied by an assignment* $m\colon \mathbf{Vars}(\phi) \to \{0,1\}$ *if* m *satisfies all constraints in* ϕ *simultaneously (such a satisfying assignment is also called a* model *of* ϕ*).*

The *weight of a model* is given by the number of variables set to true. Assuming a canonical order on the variables we can regard models as tuples in the obvious way- Also, we do not distinguish between a formula ϕ and the logical relation R_ϕ it defines, that is, the relation consisting of all models of ϕ. We will particularly consider the following constraints *implication* $\to(x,y) = (x \to y)$, *true* $\top(x) = (x)$, *false* $\bot(x) = (\bar{x})$, and *not-equal* $\mathrm{NEQ}(x,y) = (x \neq y)$.

Now we are ready to introduce the parametrised enumeration problem of interest.

Problem:	ENUM-MAXONES-SAT(Γ)
Input:	A Γ-formula ϕ and a positive integer k.
Parameter:	The integer k.
Output:	All assignments satisfying ϕ of weight $\geq k$.

Of course, a compulsory condition for the existence of a DelayFPT algorithm for ENUM-MAXONES-SAT is that its corresponding decision problem is in FPT. Let us denote this specific problem by MAXONES-SAT(Γ).

Problem:	MAXONES-SAT(Γ)
Input:	Γ-formula ϕ, integer $k \in \mathbb{N}$.
Question:	Does there exists a satisfying assignment for ϕ of weight $\geq k$?

Kratsch et al. [KMW16] studied this problem before. They completely classified the parametrised complexity of this problem in Schaefer's framework. Before we can explain their result, we need to introduce some terminology concerning types of Boolean relations which are well-known from Schaefer's original paper.

Definition 4.7 (Relational properties)**.**
We say that a Boolean relation R *is*

- a-valid *(for* $a \in \{0,1\}$*) if* $R(a,\dots,a) = 1$*,*
- Horn *(resp.,* dual Horn*) if* R *can be defined by a CNF formula which is Horn (resp., dual Horn), that is, every clause contains at most one positive (resp., negative) literal,*
- bijunctive *if* R *can be defined by a 2-CNF formula,*
- affine *if it can be defined by an* affine *formula, that is, conjunctions of* $\oplus$*-clauses (consisting of an* $\oplus$ *of some variables plus maybe the constant 1)—such a formula may also be seen as a system of linear equations over GF*[2]*, and*
- complementive *if for all* $m \in R$ *we have also* $\vec{1} \oplus m \in R$*.*

Kratsch et al. [KMW16] also introduce a new restriction of the class of bijunctive relations as follows. For this they use the notion of *frozen implementation*, stemming from the work of Nordh and Zanuttini [NZ09].

Definition 4.8 (Frozen Implementations)**.**
Let ϕ *be a formula and* $x \in \mathbf{Vars}(\phi)$*, then* x *is said to be* frozen *in* ϕ *if it is assigned the same truth value in all its models. Furthermore, we say that* Γ freezingly implements *a given relation* R *if there is a* Γ*-formula* ϕ *such that* $R(x_1,\dots x_n) \equiv \exists X \phi$*, where* ϕ *uses variables from* $X \cup \{x_1,\dots x_n\}$ *only, and all variables in* X *are frozen in* ϕ*. For the sake of readability, we let denote* $\langle\Gamma\rangle_{fr}$ *the set of all relations that can be freezingly implemented by* Γ*.*

Furthermore, we say that a relation R is *strongly bijunctive* if it is in

$$\langle\{(x \vee y),(x \neq y),(x \to y)\}\rangle_{fr}\,.$$

Definition 4.9 (Properties of constraint languages)**.**
A constraint language Γ *is dual Horn (resp., affine, strongly bijunctive,* a*-valid, complementive) if every relation in* Γ *has the respective property. We say that a constraint language is* Schaefer *if* Γ *is either Horn, dual Horn, bijunctive, or affine.*

Now we can restate the announced result of Kratsch, Marx and Wahlström.

Proposition 4.10 ([KMW16, Cor. 3.7])**.**
If Γ *is 1-valid, dual-Horn, affine, or strongly bijunctive, then* MaxOnes-SAT(Γ) *is in* FPT*. Otherwise* MaxOnes-SAT(Γ) *is* W[1]*-hard.*

Let us recall that efficient enumeration algorithms obeying DelayFPT can only exist if the underlying decision problem itself is efficiently solvable in the parametrised sense. In the following we will prove that in three of the four efficient cases from Proposition 4.10 this indeed is possible. In particular, the technique used to obtain these results will follow the paradigm of self-reducibility.

Algorithm 4.1: Enumerate the models of weight at least k

Input: A formula ϕ with $\mathbf{Vars}(\phi) = \{x_1, \ldots, x_n\}$, an integer k
Output: All satisfying assignments (given as sets of variables) of ϕ of weight $\geq k$.

```
1 if HasMaxOnes(φ, k) then Generate(φ, ∅, k, n)
```

Procedure `Generate` (CNF-Formula ϕ, Model M, Weight $w \in \mathbb{N}$, Variable index $p \in \mathbb{N}$):

```
1 if w = 0 or p = 0 then output M
2 else
3     if HasMaxOnes(φ[x_p = 1], w − 1) then Generate(φ[x_p = 1], M ∪ {x_p}, w − 1, p − 1)
4     if HasMaxOnes(φ[x_p = 0], w) then Generate(φ[x_p = 0], M, w, p − 1)
```

Theorem 4.11.
If Γ is dual-Horn, affine, or strongly bijunctive, then there is a DelayFPT *algorithm for* ENUM-MAXONES-SAT(Γ).

Proof. Algorithm 4.1 is a canonical algorithm for enumerating all satisfying assignments of weight at least k. Note that Algorithm 4.1 follows the approach of Algorithm A by Creignou and Hébrard [CH97, p. 503]. The function `HasMaxOnes`(ϕ, k) tests if the formula ϕ has a model of weight at least k. Observe that if Γ is dual-Horn, affine, or strongly bijunctive, then according to Proposition 4.10 the procedure `HasMaxOnes`(ϕ, k) performs in FPT time. Moreover, essentially, if ϕ is dual-Horn (resp., affine, strongly bijunctive) then so are $\phi[x_p = 0]$ and $\phi[x_p = 1]$ for any variable x_p. Now let us turn towards the delay estimation.

The precomputation phase of Algorithm 4.1 consists of the time required for one call to `HasMaxOnes` and consequently is in FPT for the considered cases. The procedure `Generate` then consists of several recursive calls which span a binary tree T of depth $d = \max\{k, n\}$ as follows. The root r in level 0 of T is the (first) call `Generate`$(\phi, \emptyset, k, n)$. Assume w.l.o.g. that in every level of the tree the left child consists of the call setting corresponding x_i to 0 and the right child setting x_i to 1. If j is the corresponding level of the tree then number the vertices lexicographically from 1 to 2^j and denote a specific vertex in that level by v_j^ℓ where $1 \leq j \leq 2^j$ and $0 \leq \ell \leq d$ is its level. The existence of a vertex v_j^ℓ in T for $\ell > 1$ depends on the return value of the call

$$\texttt{HasMaxOnes}(\phi[x_{p-\ell-1} = 1 - (j \mod 2)], w' - 1 + (j \mod 2)])$$

in vertex $v_{\lceil j/2 \rceil}^{\ell-1}$ where w' is the weight value of the parent of $v_{\lceil j/2 \rceil}^{\ell-1}$. If it returns true then v_j^ℓ exists. Accordingly, the recursion immediately ends if there does not exist any further solution in this branch. This observation is now used in the following claim to prove the DelayFPT-membership of the algorithm.

Claim. *The delay of Algorithm 4.1 is* FPT.

Proof of Claim. Let p be the polynomial and f be the computable function witnessing the FPT runtime of `HasMaxOnes`. Furthermore, let $T = (V, E)$ be the binary tree

depicting the recursive calls of `Generate` explained as before. In the following we have to differentiate between two cases.

Let M_{i-1}, M_i be the sets returned after, resp., before the i-th delay. Moreover, denote with $y = y_1 \cdots y_n$ and $z = z_1 \cdots z_n$ with $y_i, z_i \in \{0,1\}$ the binary vectors which describe the (assignments) M_{i-1} and M_i (that is, $z_i = 0$ if and only if $x_i \in M_i$, and $y_i = 0$ if and only if $x_i \in M_{i-1}$). The largest possible lexicographical distance between y and z is $2^n - 1$. This situation occurs exactly if $y = 0 \cdots 0$ and $z = 1 \cdots 1$ are true. More generally, if z is the lexicographic next assignment after y then the i-th delay is at most $O(n \cdot p(\phi) \cdot f(k))$. In such a case $y = 01 \cdots 1$ and $z = 10 \cdots 0$ forces the recursive calls have to ripple back to the root (case 1).

Otherwise, in case 2, there are sets $N_1, \ldots, N_m$ which are exactly the $m \in \{1, 2^n - 2\}$ lexicographic sets between M_{i-1} and M_i such that for all $1 \leq \ell \leq m$ we have that N_ℓ is not a valid solution and thus is not output. In the following we consider the size of the delay and show that it is twofold: the FPT runtime of `HasMaxOnes` times a polynomial in $|\phi|$.

If in a recursive call in level i `HasMaxOnes` denies the existence of further solutions then this excludes 2^{n-i} sets which are never considered. Accordingly, we can condense these "bad sets": let $p_1 \cdots p_r$ be the common prefix of y and z with $r \in \{0, \ldots, n-2\}$, where $r = 0$ denotes that there is no common prefix, that is, $y_1 = 0$ and $z_1 = 1$.

(1.) Now repeat the following steps for $i = n, \ldots, r+2$:
  (1.) If $y_i = 0$, then group 2^{n-i} calls of the form $y_1 \cdots y_{i-1} 1 e_1 \cdots e_{n-i}$ with $e_j \in \{0,1\}$.
  (2.) If $y_i = 1$ then do nothing.

(2.) Now we repeat the following steps for $i = r+2, \ldots, n$:
  (1.) If $z_i = 0$, then do nothing
  (2.) If $z_i = 1$, then group 2^{n-i} calls of the form $z_1 \cdots z_{i-1} 0 e_1 \cdots e_{n-i}$ with $e_j \in \{0,1\}$.

For every of these sets there is always a negatively answered `HasMaxOnes` call and thus denies further solutions in this group.

As the union of the "bad sets" in all these groups is exactly $N_1, \ldots, N_m$ we need to measure the number of such groups and thereby the runtime of the i-th delay. The maximum number of groups is created if for (1.) never (b) is fulfilled and for (2.) never (a) is fulfilled. As a result, there are at most $2 \cdot (n - r - 2)$ groups with runtimes of each in $O(p(|\phi|) \cdot f(k))$. As a consequence, together polynomially many fpt-runtimes which together is again in FPT. (Claim) ■

To conclude, the proposed enumeration algorithm is in DelayFPT. □

Now from Proposition 4.10 the efficient case for 1-valid relations is remaining. However the presented algorithm cannot work for this case as ϕ being 1-valid does not imply $\phi[x = 0]$ is 1-valid. In particular we will show a strong argument against

the possibility that this case can be efficiently solved. In Theorem 4.14 (see page 44) we will prove that ENUM-MAXONES-SAT(Γ) does not have a DelayFPT algorithm if Γ is 1-valid but neither dual-Horn nor affine nor strongly bijunctive. This result will then complete the classification of the parametrised enumeration complexity of the problem ENUM-MAXONES-SAT.

4.2.2 Enumeration of Strong HORN-Backdoor Sets

In the following we investigate the enumeration of strong backdoor sets. At first we will introduce relevant terminology which stems from the work of Williams et al. [WGS03]. Consider a formula ϕ in conjunctive normal form. Denote by $\phi[\tau]$ for a truth assignment τ the result of removing all clauses from ϕ which contain a literal ℓ with $\tau(\ell) = 1$ and removing literals ℓ with $\tau(\ell) = 0$ from the remaining clauses.

Definition 4.12 (Strong HORN-Backdoor Sets)**.**
*A set V of variables of ϕ, $V \subseteq$ **Vars**(ϕ), is a* strong HORN-backdoor set *of ϕ if for all truth assignments $\tau\colon V \to \{0,1\}$ we have $\phi[\tau] \in$ HORN.*

Observe that equivalently V is a strong HORN-backdoor set of ϕ if $\phi[V]$ is HORN, where $\phi[V]$ denotes the formula obtained from ϕ in deleting in ϕ all occurrences of variables from V.

Now let us consider the following enumeration problem.

Problem:	EXACT-STRONG-BACKDOORSET[HORN]
Input:	A formula ϕ in CNF.
Parameter:	The integer k.
Output:	The set of all strong HORN-backdoor sets of ϕ of size exactly k.

From Nishimura, Ragde, and Szeider [NRS04] we know that the detection of strong HORN-backdoor sets is in FPT (parametrised by the size of the backdoor set). Nishimura et al. use a variant of bounded-search trees in their FPT-algorithm, together with self-reducibility. This combination leads to an efficient enumeration of all strong HORN-backdoor sets of size k.

Theorem 4.13.
EXACT-STRONG-BACKDOORSET[HORN] *is in* DelayFPT.

Proof. We claim that the procedure `GenerateSBDS`(ϕ, B, k, V) depicted in Algorithm 4.2 is the required enumeration algorithm. Within its computation it accesses a function `Exists-SBDS`(ϕ, k, V). This function checks if ϕ has a strong HORN-backdoor set of size exactly k consisting of variables from V. Whenever the procedure `GenerateSBDS` outputs a set B then this set is a strong HORN-backdoor set of size k.

The function `Exists-SBDS`(ϕ, k, V) is computable in FPT [NRS04]. It uses the following fact (which is also used by Nishimura et al. to detect the existence of

Algorithm 4.2: Enumerate all strong HORN-backdoor sets of size k

Input: A CNF-formula ϕ, an integer k
Output: All strong HORN-backdoor sets of size k.

```
1  if Exists-SBDS(φ, k, Vars(φ)) then GenerateSBDS(φ, ∅, k, Vars(φ))

   Procedure GenerateSBDS (Formula φ, Backdoor B ⊆ V, Size k ∈ ℕ, Variables V):
1  if k = 0 or V = ∅ then output B
2  else
3      if Exists-SBDS(φ[B ∪ {min(V)}], k − 1, V \ {min(V)}) then
4          GenerateSBDS(φ, B ∪ {min(V)}, k − 1, V \ {min(V)})
5      if Exists-SBDS(φ[B], k, V \ {min(V)}) then GenerateSBDS(φ, B, k, V \ {min(V)})

   Function Exists-SBDS (Formula φ, Size k ∈ ℕ, Variables V):
1  if (k = 0 or V = ∅) and φ[V] ∈ HORN then return true else return false
2  if there is a clause C with two positive literals p1, p2 then
3      if exactly one of p1 and p2 is in V, say p1 ∈ V, p2 ∉ V then
4          if Exists-SBDS(φ[{p1}], k − 1, V \ {p1}) then return true
5      else
6          if p1 ∈ V and p2 ∈ V then
7              if Exists-SBDS(φ[{p1}], k − 1, V \ {p1}) then return true
8              if Exists-SBDS(φ[{p2}], k − 1, V \ {p2}) then return true
9      return false
11 else return true
```

strong HORN-backdoors [NRS04]): any non-HORN formula contains at least one non-HORN clause, that is, a clause containing at least two positive literals. W.l.o.g. let p_1 and p_2 be these two positive occurrences. Observe that then either one of these two variables must belong to any strong backdoor set of the original formula.

The enumeration algorithm is then a standard binary search recursive algorithm: in case a backdoor of size k exists, it recursively explores the strong backdoors containing the lexicographically first variable, and then the ones not containing it. The test made by the call to the function `Exists-SBDS` avoids exploring a subtree in which there is no solution to be found. As this function is in FPT, and that the depth of the binary search tree is bounded by the parameter k ensures that the enumeration algorithm has delay FPT. □

4.3 A Dichotomy for the Enumerability of Max-Ones-SAT

Conclusively, we return to the problem family ENUM-MAXONES-SAT(Γ). In the previous section, in Theorem 4.11 we exhibited three classes of logical relations Γ that allow a DelayFPT algorithm. In this section, we aim to complement this result by proving corresponding lower bounds for all remaining constraint languages Γ. As a result, we give a complete dichotomy theorem for the parametrised enumerability of ENUM-MAXONES-SAT(Γ). Remarkably, the classification of the enumeration problem

differs from the one of the decision problem (Proposition 4.10), as we state in the following theorem.

Theorem 4.14.
If Γ is dual-Horn, affine, or strongly bijunctive, then there is a DelayFPT *algorithm for* ENUM-MAXONES-SAT(Γ). *Otherwise such an algorithm does not exist unless* W[1] = FPT.

The reason why Γ being dual-Horn, affine, or strongly bijunctive implies existence of a DelayFPT algorithm for ENUM-MAXONES-SAT(Γ) is the result of Theorem 4.11. On that account, it only remains to deal with the hard cases. Notice that a DelayFPT algorithm for ENUM-MAXONES-SAT provides an FPT algorithm for the associated decision problem MAXONES-SAT. Accordingly, in order to exclude the existence of a DelayFPT algorithm (unless W[1] = FPT) for ENUM-MAXONES-SAT, it suffices to show W[1]-hardness of MAXONES-SAT. Correlating with Proposition 4.10, it suffices thus to show the W[1]-hardness of MAXONES-SAT(Γ) when Γ is 1-valid but neither dual-Horn, nor affine, nor strongly bijunctive.

To this aim we will use the problem MAXONES-SAT$^*(\Gamma)$, which, given a formula ϕ and an integer k consists in deciding whether ϕ has a nontrivial (that is, non-all-1) model of weight at least k. We will implicitly use that this parametrised problem can also be seen as a usual decision problem when neglecting the fact that k is the parameter. We will prove that MAXONES-SAT$^*(\Gamma)$ is either W[1]-hard or NP-hard otherwise. This implies that if there is a DelayFPT algorithm that enumerates all models of weight at least k of a Γ-formula, then FPT = W[1] or even, in the second case, P = NP.

From now on let us suppose that Γ is 1-valid but neither dual-Horn, nor affine, nor strongly bijunctive. Furthermore, continue proving hardness of MAXONES-SAT$^*(\Gamma)$. In the following we will use standard many-one polynomial reductions between parametrised problems (considering these problems as usual decision problems), $\leq^{\mathbf{p}}_{\mathbf{m}}$, which are *a fortiori* FPT reductions, $\leq_{\mathbf{fpt}}$. Conveniently, we will make a case distinction according to whether Γ is complementive or not.

Case 1: Γ is not complementive.

As our proof heavily relies on the notion of frozen implementation let us revisit its definition.

Definition 4.8 (Frozen Implementations)**.**
Let ϕ be a formula and $x \in$ **Vars**(ϕ)*, then x is said to be* frozen *in ϕ if it is assigned the same truth value in all its models. Furthermore, we say that* Γ freezingly implements *a given relation R if there is a Γ-formula ϕ such that $R(x_1, \ldots x_n) \equiv \exists X \phi$, where ϕ uses variables from $X \cup \{x_1, \ldots x_n\}$ only, and all variables in X are frozen in ϕ. For the sake of readability, we let denote $\langle\Gamma\rangle_{fr}$ the set of all relations that can be freezingly implemented by Γ.*

Such implementations are relevant for our purpose and we have the following:

$$\text{If } R \in \langle\Gamma\rangle_{fr}\text{, } \textsc{MaxOnes-SAT}^*(\{R\}) \leq^{\mathbf{p}}_{\mathbf{m}} \textsc{MaxOnes-SAT}^*(\Gamma). \tag{4.1}$$

Truly, the frozen implementation allows us to efficiently translate R-constraints into satisfiability-equivalent Γ-formulas with existentially quantified variables. "Freezing" of existentially quantified variables removes the quantifier and preserves the weight of the solutions. More precisely, $R \in \langle\Gamma\rangle_{fr}$ implies the existence of a Γ-formula ϕ such that $R(x_1, \dots x_n) \equiv \exists X \phi$, where ϕ uses variables from $X \cup \{x_1, \dots x_n\}$ only, and all variables in X are frozen in ϕ. Assume that variable $p \leq n$ variables are frozen to 1. On that account, given ψ is an $\{R\}$-formula, we can construct a satisfiability-equivalent Γ-formula ψ' over the variables of ψ extended by X: we replace every $\{R\}$-constraint by its satisfiability-equivalent Γ-formula and remove the existential quantifiers. Accordingly, from the pair (ψ, k) where $k \in \mathbb{N}$, we can compute the pair $(\psi', k + p)$ such that ψ has a model of weight $\geq k$ if and only if ψ' has a model of weight $\geq k + p$.

The following lemma is an immediate consequence of (4.1).

Lemma 4.15.
If there exists a relation $R \in \langle\Gamma\rangle_{fr}$ *such that* MaxOnes-SAT*(R) *is* NP*-hard, then* MaxOnes-SAT*(Γ) *is* NP*-hard as well (w.r.t.* $\leq^{\mathbf{p}}_{\mathbf{m}}$*-reductions).*

Lemma 4.16.
If Γ *is* 1*-valid but neither complementive, nor dual-Horn, nor affine, nor strongly bijunctive and* $\rightarrow \in \langle\Gamma\rangle_{fr}$*, then* MaxOnes-SAT*(Γ) *is* W[1]*-hard (w.r.t.* $\leq_{\mathbf{fpt}}$*-reductions).*

Proof. Since ⊥ is not 1-valid, $\Gamma \cup \{\bot\}$ is neither 1-valid, nor dual-Horn, nor affine, nor strongly bijunctive. As a result, according to Proposition 4.10, MaxOnes-SAT($\Gamma \cup \{\bot\}$) is W[1]-hard. In view of this, the lemma follows from the following sequence of reductions.

$$\textsc{MaxOnes-SAT}(\Gamma \cup \{\bot\}) \overset{(1)}{\leq^{\mathbf{p}}_{\mathbf{m}}} \textsc{MaxOnes-SAT}^*(\Gamma \cup \{\rightarrow\}) \overset{(2)}{\leq^{\mathbf{p}}_{\mathbf{m}}} \textsc{MaxOnes-SAT}^*(\Gamma).$$

For (1), given a $\Gamma \cup \{\bot\}$-formula ϕ over the set of variables $\{x_1, \dots, x_n\}$, let $V_\bot$ be the set of variables occurring with the constraint $\bot(x)$ in ϕ. Suppose the $\Gamma \cup \{\rightarrow\}$-formula is

$$\phi' := \phi(f/V_\bot) \wedge \bigwedge_{i=1}^{n} (f \rightarrow x_i),$$

where f is a fresh variable. Then there is a one-to-one correspondence between the models of ϕ and those of ϕ' that set f to 0. Moreover, the only model of ϕ' that sets f to 1 is the all-1 assignment. This shows that ϕ has a model of weight at least k if and only if ϕ' has one nontrivial model of weight at least k.

The reduction (2) is by virtue of (4.1) since by assumption $\rightarrow \in \langle\Gamma\rangle_{fr}$. □

In the following two lemmas we distinguish R between being 0-valid or not. This approach will show that the preconditions of either Lemma 4.15 or Lemma 4.16 eventuate. On that account we conclude the proof for a non-complementive Γ. The proof technique consist of finding relevant implementations to meet the preconditions. This is rather technical but also very standard (see Creignou et al. [CKS01]). The implementations are based on well-known characterisation of respective Horn, dual Horn, bijunctive, and affine relations that we recall (see Creignou and Vollmer [CV08] for a detailed description).

Definition 4.17 (Closure Properties)**.**
The operations of conjunction ($\wedge$), disjunction ($\vee$), majority (maj), and addition modulo 2 ($\oplus$) are applied coordinate-wisely on k-ary Boolean vectors $m, m', m'' \in \{0,1\}^k$. Given a logical relation R, the following closure properties *fully determine the structure of R:*

- R *is Horn if and only if* $m, m' \in R$ *implies* $m \wedge m' \in R$.
- R *is dual Horn if and only if* $m, m' \in R$ *implies* $m \vee m' \in R$.
- R *is bijunctive if and only if* $m, m', m'' \in R$ *implies* $maj(m, m', m'') \in R$.
- R *is affine if and only if* $m, m', m'' \in R$ *implies* $m \oplus m' \oplus m'' \in R$.
- *If R is 1-valid, then R is affine if and only if* $m, m' \in R$ *implies* $m \oplus m' \oplus (1, \ldots, 1) \in R$.

Lemma 4.18.
If Γ is 0- and 1-valid but not complementive then MaxOnes-SAT$^*(\Gamma)$ *is* W[1]*-hard (w.r.t. $\leq_{\mathbf{fpt}}$-reductions).*

Proof. Let $R \in \Gamma$ be a relation of arity m that is non-complementive (such a relation exists since by assumption Γ is non-complementive).

Consider the constraint $C = R(x_1, \ldots, x_m)$. Since R is non-complementive there exists m_1 in R such that $m_1 \oplus (1, \ldots, 1) \notin R$. For $i \in \{0,1\}$, set $V_i = \{x \in V \mid m_1(x) = i\}$. Consider the $\{R\}$-constraint defined by: $M(x,y) = C[x/V_0, y/V_1]$. This leads to the cases on the right. The first two rows follow as Γ is 0- and 1-valid. Then $M(x,y) \equiv (x \to y)$ shows that $\to \in \langle\Gamma\rangle_{fr}$ and we conclude with Lemma 4.16. □

assignment	V_0	V_1	
$\vec{1}$	1	1	$\in R$
$\vec{0}$	0	0	$\in R$
m_1	0	1	$\in R$
$m_1 \oplus \vec{1}$	1	0	$\notin R$
$M(x,y)$	x	y	

Lemma 4.19.
If Γ is 1-valid but neither 0-valid, nor complementive, nor dual-Horn, nor affine then either MaxOnes-SAT$^*(\Gamma)$ *is* W[1]*-hard (w.r.t. $\leq_{\mathbf{fpt}}$-reductions) or* MaxOnes-SAT$^*(\Gamma)$ *is* NP*-hard (w.r.t. $\leq^{\mathbf{p}}_{\mathbf{m}}$-reductions).*

Proof. As Γ is not 0-valid there exists a relation U which is 1-valid but not 0-valid. We show that the relation $\top$ can be implemented with no existential variable. Suppose

that U is of arity m, and consider the constraint $C_1 = U(x_1, \ldots, x_m)$. The unary relation defined by $C_1[x/\{x_1, \ldots, x_m\}]$ contains 1 since $U(1, \ldots, 1) = 1$, but not 0 since $U(0, \ldots, 0) = 0$. As a result, Γ can implement the relation $\top$ with no existential variable. In particular, it can freezingly implement $\top$, that is, $\top \in \langle\Gamma\rangle_{fr}$.

Now consider a non-dual-Horn relation $R \in \Gamma$ of arity m and the constraint $C = R(x_1, \ldots, x_m)$. Since R is non-dual-Horn there exist m_1 and m_2 in R such that $m_1 \vee m_2 \notin R$. For $i, j \in \{0, 1\}$, set $V_{ij} = \{ x \in V \mid m_1(x) = i \wedge m_2(x) = j \}$. Consider the $\{R\}$-constraint $M(x, y, z, t) = C[x/V_{00}, y/V_{01}, z/V_{10}, t/V_{11}]$.

This leads to the following cases:

assignment	V_{00}	V_{01}	V_{10}	V_{11}	
$\vec{1}$	1	1	1	1	$\in R$
m_1	0	0	1	1	$\in R$
m_2	0	1	0	1	$\in R$
$m_1 \vee m_2$	0	1	1	1	$\notin R$
$M(x, y, z, t)$	x	y	z	t	

Now, let $R' \in \Gamma$ be a non-affine relation of arity m' and consider the constraint $C' = R'(y_1, \ldots, y_{m'})$. As R' is non-affine and 1-valid there exist $m'_1, m'_2 \in R'$ such that $(m'_1 \oplus m'_2 \oplus (1, \ldots, 1)) \notin R'$[2]. For $i, j \in \{0, 1\}$, set

$$V'_{ij} = \{ x \in V \mid m'_1(x) = i \wedge m'_2(x) = j \}.$$

Consider the $\{R'\}$-constraint $M'(x, y, z, t) = C'[x/V'_{0,0}, y/V'_{0,1}, z/V'_{1,0}, t/V'_{1,1}]$.

Again, the following cases have to be considered:

assignment	V'_{00}	V'_{01}	V'_{10}	V'_{11}	
$\vec{1}$	1	1	1	1	$\in R'$
m'_1	0	0	1	1	$\in R'$
m'_2	0	1	0	1	$\in R'$
$m'_1 \oplus m'_2 \oplus \vec{1}$	1	0	0	1	$\notin R'$
$M'(x, y, z, t)$	x	y	z	t	

Finally consider the ternary relation Q defined by

$$Q(x, y, z) = \exists t M(x, y, z, t) \wedge M'(x, y, z, t) \wedge \top(t)$$

[2]Assume this is not true, then $\forall m, m' \in R$ s.t. $m \oplus m' \oplus (1, \ldots, 1) \in R'$ and $\exists m_1, m_2, m_3 \in R'$ s.t. $m_1 \oplus m_2 \oplus m_3 \notin R'$. But then $(m_1 \oplus m_2 \oplus (1, \ldots, 1) \oplus m_2 \oplus (1, \ldots, 1) \in R'$ which means $m_1 \oplus m_2 \oplus m_3 \in R'$ ↯.

Clearly, $Q \in \langle\Gamma\rangle_{fr}$. Moreover, the relation Q contains the following tuples.

x	y	z	tuple $\in$ Q ?
0	0	0	?
0	0	1	✓, by construction
0	1	0	✓, by construction
0	1	1	$\times$, $\notin M$
1	0	0	$\times$, $\notin M'$
1	0	1	?
1	1	0	?
1	1	1	✓, Q is 1-valid

Observe that Q is neither dual-Horn (since 001 and 010 belong to Q, but $011 = 001 \vee 010$ does not), nor affine (since 001, 010 and 111 belong to Q, but $100 = 001 \oplus 010 \oplus 111$ does not). There are three tuples for which we do not know whether they belong to Q or not, and this makes eight cases to investigate.

- If Q does not contain 000, this means that $Q = \{001, 010, 111\}$, or $Q = \{001, 010, 111, 110\}$, or $Q = \{001, 010, 111, 101\}$, or $Q = \{001, 010, 111, 110, 101\}$. In these cases it is easy to check that Q is neither Horn (for $001 \wedge 010 = 000 \notin Q$), nor bijunctive (for $\mathrm{maj}(001, 010, 111) = 011 \notin Q$). As a result Γ is not Schaefer, and according to [CH97], deciding whether a Γ-formula has a non-trivial satisfying assignment in that case is NP-hard. Accordingly, MaxOnes-SAT$^*(\Gamma)$ is NP-hard (deciding whether a formula is satisfiable reduces to deciding whether it has a model of weight at least 1), and we can conclude with Lemma 4.15.
- If $Q = \{001, 010, 111, 000\}$ or $Q = \{001, 010, 111, 000, 101\}$ then it is easy to see that $Q(x, x, y) \equiv (x \rightarrow y)$, proving that $\rightarrow \in \langle\Gamma\rangle_{fr}$. Consequently, we conclude with Lemma 4.16.
- If $Q = \{001, 010, 111, 000, 110\}$ or $Q = \{001, 010, 111, 000, 110, 101\}$ then it $\exists z Q(y, z, x) \wedge \top(z) \equiv (x \rightarrow y)$ follows again proving that $\rightarrow \in \langle\Gamma\rangle_{fr}$. Finally, we conclude with Lemma 4.16.

□

Case 2: Γ is complementive

The proof in this case will be follow the argumentation of the previous one. Nevertheless, a symmetric property is induced by this additional property. As a result, we will use a symmetric version of the implication

$$\text{Sym-Imp}(x, y, z) = (z = 0 \wedge f_{\rightarrow}(x, y)) \vee (z = 1 \wedge f_{\rightarrow}(y, x))).$$

Moreover, frozen implementations do not exist for complementive relations. That being so we will use a stronger notion of implementations.

We say that $R \in \langle\Gamma\rangle_{\exists\neq}$ if R can be implemented by a Γ-formula with no existential variables. Such implementations are relevant for our proof:

$$\text{If } R \in \langle\Gamma\rangle_{\exists\neq}, \textsc{MaxOnes-SAT}^*(\{R\}) \leq^{\mathbf{p}}_{\mathbf{m}} \textsc{MaxOnes-SAT}^*(\Gamma). \tag{4.2}$$

The following two lemmas (where the first directly follows from claim (4.2)) are used to prove the second case.

Lemma 4.20.
If there exists a relation $R \in \langle\Gamma\rangle_{\exists\neq}$ such that MaxOnes-SAT$^*(R)$ *is* NP*-hard, then* MaxOnes-SAT$^*(\Gamma)$ *is* NP*-hard as well (w.r.t. $\leq^{\mathbf{p}}_{\mathbf{m}}$-reductions).*

Lemma 4.21.
If Γ is complementive and 1*-valid but neither dual-Horn, nor affine, nor strongly bijunctive and Sym-Imp $\in \langle\Gamma\rangle_{\exists\neq}$, then* MaxOnes-SAT$^*(\Gamma)$ *is* W[1]*-hard (w.r.t. $\leq_{\mathbf{fpt}}$-reductions).*

Proof. Since NEQ is complementive but not 1-valid, $\Gamma \cup \{\text{NEQ}\}$ is neither 1-valid, nor dual-Horn, nor affine, nor strongly bijunctive. Therefore, according to Proposition 4.10, MaxOnes-SAT$(\Gamma \cup \{\neq\})$ is W[1]-hard. Hence, the lemma follows from the following sequence of reductions.

$$\begin{aligned}\textsc{MaxOnes-SAT}(\Gamma \cup \{\text{NEQ}\}) &\overset{(1)}{\leq^{\mathbf{p}}_{\mathbf{m}}} \textsc{MaxOnes-SAT}^*(\Gamma \cup \{\text{Sym-Imp}\}) \\ &\overset{(2)}{\leq^{\mathbf{p}}_{\mathbf{m}}} \textsc{MaxOnes-SAT}^*(\Gamma).\end{aligned}$$

For (1), given a $\Gamma \cup \{\neq\}$-formula ϕ over the set of variables $\{x_1, \ldots, x_n\}$, let $V_\perp$ be the set of variables occurring with the constraint $\perp(x)$ in ϕ. Now consider the $\Gamma \cup \{\rightarrow\}$-formula defined as $\phi' := \phi[f/V_\perp] \wedge \bigwedge_{i=1}^{n} \rightarrow (f, x_i)$, where f is a fresh variable. Then there is a one-to-one correspondence between the models of ϕ and those of ϕ' that set f to 0. Moreover, the only model of ϕ' that sets f to 1 is the all-1 assignment. Accordingly, ϕ has a model of weight at least k if and only if ϕ' has one nontrivial model of weight at least k. (2) follows from (4.2) since by assumption $\rightarrow \in \langle\Gamma\rangle_{\exists\neq}$. □

Lemma 4.22.
If Γ is complementive and 1*-valid but neither dual-Horn, nor affine and Sym-Imp $\in \langle\Gamma\rangle_{\exists\neq}$, then* MaxOnes-SAT$^*(\Gamma)$ *is* W[1]*-hard (w.r.t. $\leq_{\mathbf{fpt}}$-reductions) or* MaxOnes-SAT$^*(\Gamma)$ *is* NP*-hard (w.r.t. $\leq^{\mathbf{p}}_{\mathbf{m}}$-reductions).*

Proof. As every relation in Γ is complementive and 1-valid, it is also 0-valid. By assumption, there exists $R \in \Gamma$ a non-dual-Horn relation of arity m and consider the constraint $C = R(x_1, \ldots, x_m)$. As R is non-dual-Horn there exist m_1 and m_2 in R such

that $m_1 \vee m_2 \notin R$. For $i,j \in \{0,1\}$, define $V_{ij} = \{\, x \in V \mid m_1(x) = i \wedge m_2(x) = j \,\}$. Now examine the $\{R\}$-constraint: $M(x,y,z,t) = C[x/V_{0,0},\ y/V_{0,1},\ z/V_{1,0},\ t/V_{1,1}]$.

This leads to the following cases:

assignment	V_{00}	V_{01}	V_{10}	V_{11}	
$\vec{1}$	1	1	1	1	$\in R$, 1-valid
$\vec{0}$	0	0	0	0	$\in R$, 0-valid
m_1	0	0	1	1	$\in R$
$m_1 \oplus \vec{1}$	1	1	0	0	$\in R$, complementive
m_2	0	1	0	1	$\in R$
$m_2 \oplus \vec{1}$	1	0	1	0	$\in R$, complementive
$m_1 \vee m_2$	0	1	1	1	$\notin R$, non-dual-Horn
$(m_1 \vee m_2) \oplus \vec{1}$	1	0	0	0	$\notin R$, complementive
$M(x,y,z,t)$	x	y	z	t	

Now, let $R' \in \Gamma$ be a non-affine relation of arity m' and investigate the constraint $C' = R'(x_1, \ldots, x_{m'})$. As R' is non-affine and 1-valid there exist m'_1 and m'_2 in R' such that $(m'_1 \oplus m'_2 \oplus (1, \ldots, 1)) \notin R$ (for an explanation see footnote on page 52). For $i,j \in \{0,1\}$, set $V'_{ij} = \{\, x \in V \mid m'_1(x) = i \wedge m'_2(x) = j \,\}$. Examine the $\{R'\}$-constraint: $M'(x,y,z,t) = C'[x/V'_{0,0},\ y/V'_{0,1},\ z/V'_{1,0},\ t/V'_{1,1}]$.

Again, the following cases have to be considered.

assignment	V'_{00}	V'_{01}	V'_{10}	V'_{11}	
$\vec{1}$	1	1	1	1	$\in R'$, 1-valid
$\vec{0}$	0	0	0	0	$\in R'$, 0-valid
m'_1	0	0	1	1	$\in R'$
$m'_1 \oplus \vec{1}$	1	1	0	0	$\in R'$, complementive
m'_2	0	1	0	1	$\in R'$
$m'_2 \oplus \vec{1}$	1	0	1	0	$\in R'$, complementive
$m'_1 \oplus m'_2 \oplus \vec{1}$	1	0	0	1	$\notin R'$, non-affine
$m'_1 \oplus m'_2$	0	1	1	0	$\notin R'$, complementive
$M'(x,y,z,t)$	x	y	z	t	

Finally, examine the relation Q of arity four defined by the the following formula using M and M' as follows $Q(x,y,z,t) = M(x,y,z,t) \wedge M'(x,y,z,t)$. Clearly $Q \in \langle \Gamma \rangle_{\neq}$. Moreover, by construction, the relation Q contains the tuples 0011, 1100, 0101, 1010, 0000 and 1111, and neither contains 0111 nor 1000 (because of the constraint M), nor 1001, nor 0110 (because of M'). There are three pairs of tuples for which the membership in Q is not clear, namely, $(0100, 1011)$, $(0010, 1101)$ and $(0001, 1110)$, and this makes eight cases to investigate. Observe that Q is neither

dual-Horn (0011 and 0101 belong to Q, but $0111 = 0011 \vee 0101$ does not) nor affine (0011, 0101 and 0000 but $1000 = 0011 \oplus 0101 \oplus 0000$ does not).

- If Q does not contain 0001 then Q is neither Horn (for $0011 \wedge 0101 = 0001 \notin Q$), nor bijunctive (for $\mathrm{maj}(0000, 0011, 0101) = 0001 \notin Q$). As a result Γ is not Schaefer, and according to [CH97], deciding whether a Γ-formula has a non-trivial satisfying assignment is NP-hard in that case. Consequently, MaxOnes-SAT$^*(\Gamma)$ is NP-hard, and we can conclude according to Lemma 4.20.
- Case $Q = \{0000, 1111, 0011, 1100, 0101, 1010, 0001, 1110\}$ or $Q = \{0000, 1111, 0011, 1100, 0101, 1010, 0001, 1110, 0010, 1101\}$. Observe that $Q(z, x, z, y)$ implements Sym-Imp(x, y, z), and on that account proves Sym-Imp $\in \langle\Gamma\rangle_{\exists^*}$. For this reason we conclude according to Lemma 4.21.
- Case $Q = \{0000, 1111, 0011, 1100, 0101, 1010, 0001, 1110, 0100, 1011\}$ or $Q = \{0000, 1111, 0011, 1100, 0101, 1010, 0001, 1110, 0010, 1101, 0100, 1011\}$. Note that $Q(x, z, y, y)$ implements Sym-Imp(x, y, z) which proves Sym-Imp $\in \langle\Gamma\rangle_{\exists^*}$. As a result Lemma 4.21 applies.

□

Finally, we want to restate the main theorem of this chapter which follows from the previous lemmas, and afterwards close by a final remark.

Theorem 4.14.
If Γ is dual-Horn, affine, or strongly bijunctive, then there is a DelayFPT *algorithm for* Enum-MaxOnes-SAT(Γ). *Otherwise such an algorithm does not exist unless* W[1] = FPT.

It would be interesting for those cases of Γ that do not admit a DelayFPT algorithm to determine an upper bound besides the trivial exponential time bound to enumerate all solutions. In particular, are there such sets Γ for which Enum-MaxOnes-SAT(Γ) is in OutputFPT?

5 Parametrised Enumeration with Orders

> *The order that our mind imagines is like a net, or like a ladder, built to attain something.*
> *But afterward you must throw the ladder away, because you discover that, even if it was useful, it was meaningless.*
>
> **— William of Baskerville**
> "The Name of the Rose" by Umberto Eco, 1980.

In this chapter we examine how introducing orders on solutions affects the enumeration complexity. Obviously, this might increase the waiting time between output solutions compared to the case where an algorithm must not obey such an order. Interestingly, these interplays will be very visible with respect to graph modification problems. For this reason, we will introduce this kind of problem in the following section. Then, we will investigate the effects on the ordered parametrised enumeration complexity. Finally, we will reach an observation that allows reflecting on *generic* modification problems. We will obtain a suitable definition of these general problems and prove that, even in this universal problem notion, DelayFPT algorithms can be constructed via an elegant strategy. Lastly, we show how this strategy can be exemplified on string and logic related problems.

5.1 Graph Modification Problems

Graph modification problems have been studied for a long time in computational complexity theory. Already in the monograph by Garey and Johnson [GJ90], among the graph-theoretic problems considered, many fall into this problem class. To the best of our knowledge, graph modification problems were studied in the context of parametrised complexity for the first time by Cai [Cai96]. A *graph property* $\mathcal{P} \subseteq \mathcal{G}$ is a set of graphs. Given some graph property $\mathcal{P}$ and some graph G, we write $G \models \mathcal{P}$ if the graph G obeys the property $\mathcal{P}$, that is, $G \in \mathcal{P}$.

In the following, we will define so-called graph operations, e.g., "removing a vertex", or "adding/removing an edge". Furthermore, we need to explicitly say what "dependence" and "consistency" of such operations means.

Definition 5.1 (Operations, Consistency, Dependency)**.**
A (graph) operation *for* G *is always with respect to a vertex or an edge:*

- *remove a vertex: this operation is a function* $\mathrm{rem}_v \colon \mathcal{G} \to \mathcal{G}$ *such that* $\mathrm{rem}_v(G)$ *is the graph obtained by removing the vertex* v *from* G *(if* v *is present; otherwise* rem_v *is the identity) and deleting all incident edges to* v*,*
- *add/remove an edge: this operation is a function* $\mathrm{add}_{\{u,v\}}, \mathrm{rem}_{\{u,v\}} \colon \mathcal{G} \to \mathcal{G}$ *such that* $\mathrm{add}_{\{u,v\}}(G)$*,* $\mathrm{rem}_{\{u,v\}}(G)$ *is the graph obtained by adding/removing the edge* $\{u, v\}$ *to* G *if* u *and* v *are present in* G*; otherwise both functions are the identity.*

Two graph operations o, o' *are* dependent *if*

- $o = \mathrm{rem}_v$ *and* $o' = \mathrm{rem}_{\{u,v\}}$ *(*o *removes the vertex* v *and* o' *removes an edge incident to* v*), or*
- $o = \mathrm{rem}_{\{u,v\}}$ *and* $o' = \mathrm{add}_{\{u,v\}}$ *(*o *removes the edge* $\{u, v\}$ *and* o' *adds the same edge* $\{u, v\}$ *again).*

A set of graph operations is consistent *if it does not contain two dependent operations. Given such a consistent set of operations* S*, the graph obtained from* G *by applying the operations in* S *on* G *is denoted by* $S(G)$*.*

The following definition introduces solutions by speaking about minimality in terms of the $\subseteq$-relation on such sets.

Definition 5.2 (Solutions)**.**
Given some graph property $\mathcal{P}$*, a graph* G*,* $k \in \mathbb{N}$*, and a set of operations* O*, we say that* S *is a* solution for (G, k, O) with respect to $\mathcal{P}$ *if the following three properties are fulfilled:*

(1.) $S \subseteq O$ *is a consistent set of operations.*

(2.) $|S| \leq k$*.*

(3.) $S(G) \models \mathcal{P}$*.*

A solution S *is* minimal *if there is no solution* S' *such that* $S' \subsetneq S$*.*

Notice that (1.) of the previous definition might be algorithmically very relevant. Enumeration algorithms that construct solutions iteratively often require to verify that the consecutive "super"-solution (in the $\subseteq$-sense) stays consistent. As a result, we will only consider solutions where this check is easy, that is, it can be achieved in polynomial (or FPT) time.

Cai [Cai96] was interested in the following parametrised graph modification decision problem w.r.t. some given graph property $\mathcal{P}$:

Problem:	$\mathcal{M}_\mathcal{P}$ — the (parametrised) graph modification problem
Input:	G undirected graph, $k \in \mathbb{N}$, O set of operations on G.
Parameter:	The integer k.
Question:	Exists a solution for (G, k, O) with respect to $\mathcal{P}$?

Cai examined the parametrised complexity of $\mathcal{M}_\mathcal{P}$ and obtained a positive result. However, before we can state his result, we have to introduce some graph terminology. Given two graphs $G = (V, E)$ and $H = (V', E')$, we write $H \leq G$ if H is an induced subgraph of G, i. e., $V' \subseteq V$ and $E' = E \cap (V' \times V')$.

Definition 5.3 (Finite Forbidden Set Characterisation)**.**
Let $\mathcal{G}$ be a finite set of graphs and $\mathcal{P}$ be some graph property. We say that $\mathcal{P}$ has a finite forbidden set characterization *$\mathcal{G}$ if for any graph G we have that $G \models \mathcal{P}$ if and only if for all $H \in \mathcal{G}$ we have $H \not\leq G$.*

In his proof, Cai developed an algorithm for the decision problem using a bounded search tree whose exhaustive exploration produces all minimal solutions.

Proposition 5.4 ([Cai96])**.**
The problem $\mathcal{M}_{\mathcal{P}}$ is in FPT *for any property $\mathcal{P}$ that has a finite forbidden set characterisation.*

A selection of significant graph modification problems is presented in the following paragraphs.

A *chord* in a graph $G = (V, E)$ is an edge between two vertices of a cycle C in G which is not part of C. A given graph $G = (V, E)$ is *triangular* (or *chordal*) if each of its induced cycles of 4 or more nodes has a chord.

Problem:	TRIANGULATION
Input:	undirected Graph $G = (V, E)$, $k \in \mathbb{N}$, set of operations $O \subseteq \{ \mathrm{add}_{\{u,v\}} \mid u, v \in V, \{u, v\} \notin E \}$.
Parameter:	The integer k.
Question:	Exists a solution of cardinality $\leq k$ such that $S(G)$ is triangular?

Proposition 5.5 ([Yan81])**.**
The decision variant of TRIANGULATION *is* NP*-complete w.r.t. $\leq^{\mathbf{p}}_{\mathbf{m}}$-reductions.*

Proposition 5.6 ([Cai96; KST99])**.**
TRIANGULATION *is in* FPT.

In the sense of Definition 5.2, a solution of TRIANGULATION then is a set of edges which have to be added to the graph to make the graph triangular.

Observe that in this special case of the modification problem, the underlying property $\mathcal{P}$, "to be triangular", does not have a finite forbidden set characterisation (since cycles of any length can be problematic). Nevertheless, one can efficiently enumerate all minimal solutions as we will see later in Corollary 5.11.

A *cluster* is a graph such that all its connected components are cliques. To transform (or modify) a graph G we restrict the allowed operations to: adding or removing an edge.

Problem:	CLUSTER-EDITING
Input:	undirected Graph $G = (V, E)$, $k \in \mathbb{N}$, set of operations $O \subseteq \{\, \mathrm{add}_{\{u,v\}} \mid u, v \in V \,\} \cup \{\, \mathrm{rem}_{\{u,v\}} \mid \{u, v\} \in E \,\}$.
Parameter:	The integer k.
Question:	Exists a solution S of cardinality $\leq k$ such that $S(G)$ is a cluster?

For CLUSTER-EDITING the set of forbidden subgraphs are paths of length two.

Proposition 5.7 ([SST04])**.**
The decision variant of CLUSTER-EDITING *is* NP*-complete w.r.t.* $\leq^{\mathbf{p}}_{\mathbf{m}}$*-reductions.*

Problem:	TRIANGLE-DELETION
Input:	undirected graph $G = (V, E)$, $k \in \mathbb{N}$, set of operations $O \subseteq \{\, \mathrm{rem}_v \mid v \in V \,\}$.
Parameter:	The integer k.
Question:	Exists a solution S of cardinality $\leq k$ such that $S(G)$ is triangle-free?

Proposition 5.8 ([Yan78])**.**
The decision variant of TRIANGLE-DELETION *is* NP*-complete w.r.t.* $\leq^{\mathbf{p}}_{\mathbf{m}}$*-reductions.*

In this case, the forbidden patterns are obviously just triangles. Further analogous problems can be defined for many other classes of graphs such as line graphs, claw-free graphs, Helly circular-arc graphs, etc., see Brandstädt et al. [BLS88].

In the following, we are interested in the corresponding enumeration problems with particular orders. Specifically, we will concentrate on two well-known preorders: lexicographic and by size. Observe that solutions in our setting are subsets of an ordered set of operations. As a result, they can be encoded as binary strings in which the ith bit from right indicates whether the ith operation is in the subset. Furthermore, we define the *lexicographic order* of solutions as the lexicographic order of these strings. The *size* of a solution simply is its cardinality. The two variants of graph modification enumeration problems are now defined.

Problem:	ENUM-$\mathcal{M}^{\mathrm{LEX}}_{\mathcal{P}}$
Input:	(G, k, O), G undirected graph, $k \in \mathbb{N}$, O ordered set of operations on G.
Parameter:	The integer k.
Output:	All solutions of (G, k, O) w.r.t. $\mathcal{P}$ in lexicographic order.

Problem:	ENUM-$\mathcal{M}_{\mathcal{P}}^{\text{SIZE}}$
Input:	(G, k, O), G undirected graph, $k \in \mathbb{N}$, O set of operations on G.
Parameter:	The integer k.
Output:	All solutions of (G, k, O) w.r.t. $\mathcal{P}$ in non-decreasing size.

If ambiguity is impossible we omit the subscript $\mathcal{P}$ for the graph modification problem and simply write $\mathcal{M}$. Analogously, we write $\text{Sol}_{\mathcal{M}}(x)$ for the function associating solutions to a given instance, and also $\mathcal{S}_{\mathcal{M}}$ for the set of all solutions of $\mathcal{M}$.

5.1.1 Lexicographic Order

The first result in this subsection will build on the parametrised complexity of the corresponding graph modification problem $\mathcal{M}_{\mathcal{P}}$. If this problem is fixed-parameter tractable then there exists an efficient enumeration algorithm for ENUM-$\mathcal{M}_{\mathcal{P}}^{\text{LEX}}$.

Theorem 5.9.
Let $\mathcal{M}_{\mathcal{P}}$ be a graph modification problem. If $\mathcal{M}_{\mathcal{P}}$ is in FPT *then* ENUM-$\mathcal{M}_{\mathcal{P}}^{\text{LEX}} \in$ DelayFPT *with polynomial space.*

Proof. Algorithm 5.1 enumerates all solutions of an instance of a given modification problem $\mathcal{M}_{\mathcal{P}}$ by the method of self-reducibility (this is an extension of the flash light search of Creignou and Hébrard [CH97]). The algorithm uses a function $\texttt{ExistsSol}(G, k, O)$ that tests if the instance (G, k, O) of the modification problem $\mathcal{M}_{\mathcal{P}}$ has a solution. By precondition of the theorem, this test is in FPT. We use calls to this function to avoid exploration of branches of the recursion tree that do not lead to any output. Ensuring that the solutions containing o_p are consistent can be done in polynomial time for graph operations. From this, we get a search tree of depth at most k. As a result, for any instance of length n, the time between the output of any two solutions is bounded by $f(k) \cdot p(n)$ for some polynomial p and an arbitrary recursive function f. □

Recall (see page 59) that TRIANGLE-DELETION and CLUSTER-EDITING possess a finite forbidden set characterisation whereas TRIANGULATION does not. As problems with these characterisations are fixed-parameter tractable (Proposition 5.4), by virtue of Theorem 5.9 there exists a delay- and space-efficient parametrised enumeration algorithm for the lexicographic order.

Proposition 5.10.
For any graph modification problem $\mathcal{M}_{\mathcal{P}}$, if $\mathcal{P}$ has a finite forbidden set characterisation then ENUM-$\mathcal{M}_{\mathcal{P}}^{\text{LEX}} \in$ DelayFPT *with polynomial space.*

Algorithm 5.1: Enumerate all solutions of $\mathcal{M}_{\mathcal{P}}$ in lexicographic order

Input: a graph G, $k \in \mathbb{N}$, an ordered set of operations $O = \{o_1, \ldots, o_n\}$
Output: all consistent sets $S \subseteq O$ s.t. $|S| \leq k$, $S(G) \models \mathcal{P}$ in lexicographic order

```
1 if ExistsSol(G, k, O) then Generate(G, k, O, ∅)

Procedure Generate(G, k, O, S):
1 if O = ∅ or k = 0 then return S
2 else
3     let o_p be the lexicographically last operation in O, let O := O \ {o_p}
4     if ExistsSol(S(G), k, O) then Generate(S(G), k, O, S)
5     if S ∪ {o_p} is consistent and ExistsSol((S ∪ {o_p})(G), k − 1, O) then
6         Generate((S ∪ {o_p})(G), k − 1, O, S ∪ {o_p})
```

Despite of lacking such a fruitful characterisation, the following corollary shows that enumerating all solutions in lexicographic order is still possible for Triangle-Deletion.

Corollary 5.11.
Enum-Triangulation$^{\text{LEX}} \in$ DelayFPT *with polynomial space.*

Proof. From Proposition 5.6 we know that Triangulation $\in$ FPT. Now apply Theorem 5.9 and receive the result. □

5.1.2 Order by Size

A neighbourhood function mapping solutions to "close" solutions is a common routine in enumeration contexts and usually allows to efficiently compute the next elements [AF96]. Essentially, we will define a specific kind of such a function which generates from a given *seed* the *initial solutions*. These are then used to produce all remaining solutions. In a sense, initial solutions can be seen as some kind of *minimal solutions*. Later in this subsection, we will see that this observation can also be taken literally.

After introducing the formal definition of neighbourhood functions, we show how to utilise priority queues leads to a general DelayFPT scheme. In the following $\mathbb{O}$ (the "seed") is a technical symbol that will be used in order to generate the initial solutions.

Definition 5.12 (Neighbourhood function)**.**
Let $\mathcal{M}$ be some graph modification problem, $I_{\mathcal{M}}$ be the set of instances, $\mathcal{S}_{\mathcal{M}}$ be the set of all solutions, and $\mathrm{Sol}_{\mathcal{M}} \colon I_{\mathcal{M}} \to \mathcal{S}_{\mathcal{M}}$ be the solution function. A neighbourhood function *for $\mathcal{M}$ is a (partial) function*

$$\mathcal{N}_{\mathcal{M}} \colon I_{\mathcal{M}} \times (\mathcal{S}_{\mathcal{M}} \cup \{\mathbb{O}\}) \to 2^{\mathcal{S}_{\mathcal{M}}}$$

such that the following is fulfilled:
(1.) For all $x = (G, k, O) \in I_{\mathcal{M}}$ and $S \in \mathrm{Sol}_{\mathcal{M}}(x) \cup \{\mathbb{O}\}$, $\mathcal{N}_{\mathcal{M}}(x, S)$ is defined.

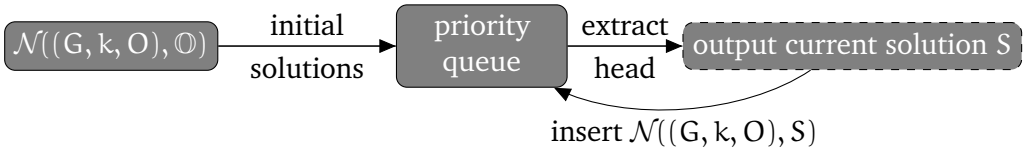


Fig. 5.1: General structure of Algorithm 5.2.

(2.) For all $x \in I_{\mathcal{M}}$, $\mathcal{N}_{\mathcal{M}}(x,\mathbb{O}) = \emptyset$ *if* $\mathrm{Sol}_{\mathcal{M}}(x) = \emptyset$, *and* $\mathcal{N}_{\mathcal{M}}(x,\mathbb{O})$ *is an arbitrary set of solutions otherwise.*

(3.) For all $x \in I_{\mathcal{M}}$ *and* $S \in \mathrm{Sol}_{\mathcal{M}}(x)$, *if* $S' \in \mathcal{N}_{\mathcal{M}}(x,S)$ *then* $|S| < |S'|$.

(4.) For all $x \in I_{\mathcal{M}}$ *and all* $S \in \mathrm{Sol}_{\mathcal{M}}(x)$, *there exists* $p > 0$ *and* $S_1,\ldots,S_p \in \mathrm{Sol}_{\mathcal{M}}(x)$ *such that*

(i) $S_1 \in \mathcal{N}_{\mathcal{M}}(x,\mathbb{O})$,

(ii) $S_{i+1} \in \mathcal{N}_{\mathcal{M}}(x,S_i)$ *for* $1 \leq i < p$, *and*

(iii) $S_p = S$.

Furthermore, we say that $\mathcal{N}_{\mathcal{M}}$ *is* FPT*-computable, when* $\mathcal{N}_{\mathcal{M}}(x,S)$ *is computable in time* $f(k)\cdot poly(|x|)$ *for any* $x \in I_{\mathcal{M}}$, $S \in \mathrm{Sol}_{\mathcal{M}}(x)$, k *is the parameter of input* x *and a recursive function* f.

Crucially, a neighbourhood function for a problem $\mathcal{M}$ is a function that provides from scratch some initial set of solutions (see Definition 5.12 (2.)). In several of the presented applications below, this initially produced set $\mathcal{N}_{\mathcal{M}}(x,\mathbb{O})$ will be encompass all minimal solutions for x. Then, from these solutions one iteratively calculates the remaining solutions (see condition (3.)), where condition (4.) guarantees that we do not miss any solution. This scheme is the key procedure in the proof of the next theorem.

Theorem 5.13.
Let $\mathcal{M}_{\mathcal{P}}$ *be a graph modification problem. If* $\mathcal{M}_{\mathcal{P}}$ *admits a neighbourhood function* $\mathcal{N}_{\mathcal{M}_{\mathcal{P}}}$ *that is* FPT*-computable, then* ENUM-$\mathcal{M}_{\mathcal{P}}^{\mathrm{SIZE}} \in$ DelayFPT *with exponential space.*

Proof. We claim that Algorithm 5.2 outputs all solutions in DelayFPT. A graphical representation of the algorithm is shown in Figure 5.1. Recall that inserting an element into the priority queue is only executed if this element is not present yet. Furthermore, the cardinality of elements of $\mathcal{N}_{\mathcal{M}_{\mathcal{P}}}((G,k,O),S)$ strictly increases compared to the cardinality of S by Definition 5.12 (3.). As a result, the output solutions obey the order (solutions with smaller cardinality are output before larger ones) and no solution is output twice (due to the priority queue combined with the size argument).

Now we will show that no solution is omitted during the computation. If $S \in \mathrm{Sol}_{\mathcal{M}_{\mathcal{P}}}(G,k,O)$ is a valid solution and $S_1,\ldots,S_p$ associated with S by Definition 5.12 (4.), we prove by induction that each S_i will appear in Q during the enumeration:

IB: consider $i = 1$ and the claim follows then from line 2 of the algorithm.

IS: consider $i > 1$. Then solution S_{i-1} is inserted in Q by the induction hypothesis and accordingly all elements of $\mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),S_{i-1})$, and in particular $S_i \in \mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),S_{i-1})$, are added to Q (line 5). As a result, S_i, and also $S = S_p$, eventually are output at a later step.

Finally, we claim that the delay of Algorithm 5.2 is FPT. Note that between the output of two consecutive solutions the runtime is bounded by two factors. Firstly, the time required to compute a neighbourhood of the form $\mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),\mathbb{O})$ or $\mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),S)$. Secondly, the time to insert all its elements in the priority queue. This is in FPT due to the assumption on $\mathcal{N}_{\mathcal{M}_\mathcal{P}}$ being FPT-computable and as there is only a single extraction and FPT-many insertion operations on the queue. □

Algorithm 5.2: DelayFPT algorithm for ENUM-$\mathcal{M}$

Input : (G,k,O), G is an undirected graph, $k \in \mathbb{N}$, and O is a set of operations.
1 compute $\mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),\mathbb{O})$;
2 insert all elements of $\mathcal{N}_{\mathcal{M}}((G,k,O),\mathbb{O})$ into priority queue Q (ordered by size);
3 **while** Q is not empty **do**
4 **extract** the minimum solution S of Q and output it;
5 **insert** all elements of $\mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),S)$ into Q;

Naturally, one can interpret the *inclusion minimal* solutions of a considered problem $\mathcal{M}_\mathcal{P}$ as an initial way to define a neighbourhood function. Now we consider the following problem.

Problem:	ENUM-MIN-$\mathcal{M}_\mathcal{P}$
Input:	G undirected graph, $k \in \mathbb{N}$, O set of operations on G.
Parameter:	The integer k.
Output:	All inclusion minimal solutions of $\mathcal{M}_\mathcal{P}$.

Theorem 5.14.
Let $\mathcal{M}_\mathcal{P}$ be a graph modification problem. If ENUM-MIN-$\mathcal{M}_\mathcal{P}$ *is* FPT*-enumerable then* ENUM-$\mathcal{M}_\mathcal{P}^{\text{SIZE}} \in$ DelayFPT *with exponential space.*

Proof. Let $\mathcal{A}$ be an FPT-algorithm for ENUM-MIN-$\mathcal{M}_\mathcal{P}$. By Theorem 5.13, it suffices to build an neighbourhood function for $\mathcal{M}_\mathcal{P}$ which is computable in FPT time. Given an instance (G,k,O) of $\mathcal{M}_\mathcal{P}$ and for $S \in \text{Sol}_{\mathcal{M}_\mathcal{P}}(G,k,O) \cup \{\mathbb{O}\}$, we define $\mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),S)$ as the result of Algorithm 5.3.

We will show that the function $\mathcal{N}_{\mathcal{M}_\mathcal{P}}$ fulfils conditions (2.) and (3.) of Definition 5.12. We prove by induction that it also satisfies condition (4.) (that is, each solution T of size k comes with a sequence $T_1,\ldots,T_p = T$ such that $T_1 \in \mathcal{N}_{\mathcal{M}}((G,k,O),\mathbb{O})$ and $T_{i+1} \in \mathcal{N}_{\mathcal{M}}((G,k,O),T_i)$ for each i).

If T is a minimal solution for (G,k,O), then $T \in \mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),\mathbb{O})$ and the expected sequence (T_i) reduces to $T_1 = T$. Otherwise, there is an $S \in \text{Sol}_{\mathcal{M}_\mathcal{P}}(G,k,O)$ and a non-empty set of transformations, say $S' \cup \{t\}$, such that $T = S \cup S' \cup \{t\}$ and there

Algorithm 5.3: Procedure for computing $\mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),S)$

```
   Input  : (G,k,O),S  G is an undirected graph, k ∈ ℕ, and O,S are sets of operations.
1  if S = ℂ then return 𝒜(G,k,O);
2  res := ∅ ;
3  forall the t ∈ O do
4  |   forall the S′ ∈ 𝒜((S ∪ {t})(G), k − |S| − 1, O \ {t}) do
5  |   |   if S ∪ S′ ∪ {t} is consistent then  res := res ∪ {S ∪ S′ ∪ {t}} ;
6  return res;
```

is no solution for G between S and $S \cup S' \cup \{t\}$. This entails that S' is a minimal solution for $((S \cup \{t\})(G), k - |S| - 1)$ and accordingly $T \in \mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),S)$ (see lines 4–5 of Algorithm 5.3). The conclusion follows from the induction hypothesis that guarantees the existence of solutions $S_1, \ldots, S_q$ such that $S_1 \in \mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),\mathbb{O})$, $S_{i+1} \in \mathcal{N}_{\mathcal{M}_\mathcal{P}}((G,k,O),S_i)$ and $S_q = S$. The expected sequence $T_1, \ldots, T_p$ for T is nothing but $S_1, \ldots, S_q, T$. To conclude, it remains to see that Algorithm 5.3 is in FPT. This follows from the fact that $\mathcal{A}$ is an FPT-algorithm (lines 1 and 4). Of course, as required before on page 58, the consistency of solutions (line 5) has to be checkable in polynomial (or FPT) time. □

Corollary 5.15.
ENUM-TRIANGULATION$^{\text{SIZE}}$ ∈ DelayFPT *with exponential space.*

Proof. All inclusion-minimal k-triangulations can be output in time $O(2^{4k} \cdot |E|)$ for a given graph G and $k \in \mathbb{N}$ [KST99, Thm. 2.4]. This immediately yields the expected result, by help of Theorem 5.14. □

Corollary 5.16.
For any property $\mathcal{P}$ that has a finite forbidden set characterisation, the problem ENUM-$\mathcal{M}_\mathcal{P}^{\text{SIZE}}$ *is in* DelayFPT.

Proof. The algorithm developed by Cai [Cai96] for the decision problem is based on a bounded search tree, whose exhaustive examination provides all minimal solutions in FPT. Theorem 5.14 yields the conclusion. □

Corollary 5.17.
ENUM-CLUSTER-EDITING$^{\text{SIZE}}$ *and* ENUM-TRIANGLE-DELETION$^{\text{SIZE}}$ *are in* DelayFPT *with exponential space.*

Proof. Both properties have a finite forbidden set characterisation, namely, paths of length two and triangles. Corollary 5.16 then proves the corollary. □

5.2 Generalised Modification Problems

In this section we show how the previous approach can be completely generalised and applied to various other structures than graphs. Subsequently, we demonstrate

the new technique for two other kinds of problems whose inputs are on the one hand strings and on the other formulas.

Definition 5.18 (General Operations)**.**
Let $Q \subseteq \Sigma^*$ *be some language defined over an alphabet, and* $x \in \Sigma^*$ *be an input. A* set of operations $\Omega = \{\, \omega_n \colon \Sigma^* \to \Sigma^* \mid n \in \mathbb{N} \,\}$ *is an infinite set of operations on instances of* Q *(for instance, adding some edges, or flipping some bit). An operation* ω *is* valid *with respect to an instance* $x \in Q$*, if* $\omega(x) \in Q$*. Write* Ω/x *for the set of possible (valid) operations on an instance* x*.*

Two operations ω, ω' *are* dependent *with respect to an instance* $x \in Q$ *if*

- $\omega(\omega'(x)) = x$*, or* *(*ω *and* ω' *cancel out)*
- $\omega(\omega'(x)) = \omega'(x)$ *or* $\omega(\omega'(x)) = \omega(x)$*.* *(order of* ω, ω' *is irrelevant)*

A set of operations $O \subseteq \Omega/x$ *is* consistent *with respect to* x *if it does not contain two dependent operations. Similarly, we say that an operation* ω *is* consistent with a set S *if and only if* $S \cup \{\omega\}$ *is consistent.*

Example 5.19. *Let* $\mathcal{G} \subseteq \{0,1\}^*$ *be the language of all undirected graphs encoded by adjacency matrices. Then* $\Omega(\mathcal{G})$ *is the set of all graph operations in the sense of Definition 5.1: removing vertices or edges, adding edges. Note that* Ω *contains all operations of the kind*

$$\mathrm{rem}_i, \quad \mathrm{rem}_{\{i,j\}}, \quad \mathrm{add}_{\{i,j\}}$$

for all $i, j \in \mathbb{N}$*. Furthermore, let* $G = (V, E) \in \{0,1\}^*$ *be a concrete input graph. That being so,* Ω/G *then is the restriction of* Ω *to those* $i, j \in \mathbb{N}$ *such that* $i, j \in V$ *encode vertices in* G*.*

Definition 5.20 (General Solutions)**.**
Let $Q \subseteq \Sigma*$ *be a language over* Σ*. We say that* S *is a* solution (of x) *if* S *is a consistent set of operations and* $S(x) \in \mathcal{P}$*. If* S *is a consistent set of operations then we write* $S(x)$ *for the application of the operations in* S *to* x*. Furthermore, we denote by* $\mathcal{S}_Q := \bigcup_{x \in Q} \{\, S \mid S \text{ is a solution of } x \,\}$ *the set of all solutions for every instance* $x \in Q$*. Also* $\mathrm{Sol}(x)$ *is the set of solutions for every instance* $x \in Q$*.*

Example 5.21. *Continuing the previous example, if the property* $\mathcal{P}$ *is "to be a cluster" then a consistent solution* S *to a given graph just then is a sequence of removing vertices, adding and deleting of edges where*

- *there is no edge* (i, j) *added or deleted such that vertex* i *or* j *is removed,*
- *there is no edge* (i, j) *added and removed, and (of course)*
- $S(G) \models \mathcal{P}$*.*

Similarly, adding edge (i, j) *together with removing vertex* i *or* j *or removing edge* (i, j) *is an inconsistent set of operations.*

Now we want to define the corresponding decision and enumeration tasks. On that account let $\mathcal{P}$ be some property, $\Pi = (Q, \kappa)$ some parametrised problem with $Q \subseteq \Sigma^*$, and Ω be a set of operations.

Problem:	$\Pi_{\mathcal{P}}$ — parameterised modification problem Π w.r.t. a property $\mathcal{P}$ over Σ
Input:	$x \in \Sigma^*$, $k \in \mathbb{N}$, Ω/x set of operations.
Parameter:	The integer $k \in \mathbb{N}$.
Question:	Is there a consistent solution $S \subseteq \Omega/x$ and $\lvert S\rvert \leq k$?

Problem:	ENUM-MIN-$\Pi_{\mathcal{P}}$ — parameterised minimum enumeration modification problem w.r.t. a property $\mathcal{P}$ over Σ
Input:	$x \in \Sigma^*$, $k \in \mathbb{N}$, Ω/x set of operations.
Parameter:	The integer $k \in \mathbb{N}$.
Output:	All minimal (w.r.t. some order) solutions $S \subseteq \Omega/x$ with $\lvert S\rvert \leq k$.

The enumeration modification problem where we want to output all possible sets of transformations on a given instance x (and not only the minimum ones) then is ENUM-$\Pi_{\mathcal{P}}$.

Next, we present the generalised notion of neighbourhood functions. Utilising this will yield generalisations of the results for graph modification problems afterwards.

Definition 5.22.
Let Σ be an alphabet, $\mathcal{P} \subseteq \Sigma^*$ be a property and $\Pi_{\mathcal{P}}$ be a parameterised modification problem over Σ. A neighbourhood function *for $\Pi_{\mathcal{P}}$ is a (partial) function $\mathcal{N}_{\Pi_{\mathcal{P}}} \colon \Sigma^* \times (\mathcal{S}_{\Pi_{\mathcal{P}}} \cup \{\mathbb{O}\}) \to 2^{\mathcal{S}_{\Pi_{\mathcal{P}}}}$ such that the following four properties are fulfilled.*

(1.) For all $x \in \Sigma^*$ and $S \in \mathrm{Sol}_{\Pi_{\mathcal{P}}}(x) \cup \{\mathbb{O}\}$, $\mathcal{N}_{\Pi_{\mathcal{P}}}(x, S)$ is defined.

(2.) For all $x \in \Sigma^*$, $\mathcal{N}_{\Pi_{\mathcal{P}}}(x, \mathbb{O}) = \emptyset$ if $\mathrm{Sol}_{\Pi_{\mathcal{P}}}(x) = \emptyset$, and $\mathcal{N}_{\Pi_{\mathcal{P}}}(x, \mathbb{O})$ is an arbitrary set of solutions otherwise.

(3.) For all $x \in \Sigma^*$ and $S \in \mathrm{Sol}_{\Pi_{\mathcal{P}}}(x)$, if $S' \in \mathcal{N}_{\Pi_{\mathcal{P}}}(x, S)$ then $\lvert S\rvert < \lvert S'\rvert$.

(4.) For all $x \in \Sigma^*$ and all $S \in \mathrm{Sol}_{\Pi_{\mathcal{P}}}(x)$, there exists $p > 0$ and $S_1, \ldots, S_p \in \mathrm{Sol}_{\Pi_{\mathcal{P}}}(x)$ such that

(i) $S_1 \in \mathcal{N}_{\Pi_{\mathcal{P}}}(x, \mathbb{O})$,

(ii) $S_{i+1} \in \mathcal{N}_{\Pi_{\mathcal{P}}}(x, S_i)$ for $1 \leq i < p$, and

(iii) $S_p = S$.

Furthermore, we say that $\mathcal{N}_{\Pi_{\mathcal{P}}}$ is FPT*-computable, when $\mathcal{N}_{\Pi_{\mathcal{P}}}(x, S)$ is computable in time $f(k) \cdot poly(\lvert x\rvert)$ for any $x \in \Sigma^*$ and $S \in \mathrm{Sol}_{\Pi_{\mathcal{P}}}(x)$.*

Finally, we are able to state generalised versions of Theorems 5.13 and 5.14.

Corollary 5.23.
Let $\mathcal{P}$ be some property, $\Pi \subseteq \Sigma^* \times \mathbb{N}$ be some parameterised modification problem,

and Ω be a set of operations such that Ω/x is finite for all $x \in \Sigma^*$. If $\Pi_{\mathcal{P}}$ admits a neighbourhood function that is FPT*-computable then* ENUM-$\Pi_{\mathcal{P}} \in$ DelayFPT *and*

- *polynomial space for lexicographic order, and*
- *exponential space for size order.*

Corollary 5.24.
Let $\mathcal{P}$ be some property, $\Pi \subseteq \Sigma^* \times \mathbb{N}$ be some parameterised modification problem, and Ω be a set of operations such that Ω/x is finite for all $x \in \Sigma^*$. If ENUM-MIN-$\Pi_{\mathcal{P}}$ *is* FPT*-enumerable and consistency of solutions can be checked in* FPT *then* ENUM-$\Pi_{\mathcal{P}} \in$ DelayFPT *and*

- *polynomial space for lexicographic order, and*
- *exponential space for size order.*

5.2.1 Closest String

In the following we consider a significant problem in coding theory [FL97]. Given a set of binary strings I we want to find a string s whose maximum Hamming distance $\max\{ d_H(s, s') \mid s' \in I \} \leq d$ for some $d \in \mathbb{N}$. This problem is NP-complete with respect to $\leq^{\mathbf{p}}_{\mathbf{m}}$-reductions.

Definition 5.25 (Bit-flip operation)**.**
Given a string $w = w_1 \cdots w_n$ with $w_i \in \{0,1\}, n \in \mathbb{N}$, and a set $S \subseteq \{1, \ldots, n\}$, $S(w)$ denotes the string obtained from w in flipping the bits indicated by S, more formally $S(w) := S(w_1) \cdots S(w_n)$, where $S(w_i) = 1 - w_i$ if $i \in S$ and $S(w_i) = w_i$ otherwise.

The corresponding parametrised version is the following.

Problem:	CLOSEST-STRING
Input:	$(s_1, \ldots, s_k, n, d)$, where $s_1, \ldots, s_k$ is a sequence of strings over $\{0,1\}$ of length $n \in \mathbb{N}$, $d \in \mathbb{N}$.
Parameter:	The integer d.
Question:	Does there exist $S \subseteq \{1, \ldots, n\}$ such that $d_H(S(s_1), s_i) \leq d$ for all $1 \leq i \leq k$??

Proposition 5.26 ([GNR03])**.**
CLOSEST-STRING *is in* FPT.

Moreover, an exhaustive examination of a bounded search tree constructed from the idea of Gramm et al. [GNR03, Fig. 1] allows to produce all minimal solutions of this problem in FPT. Accordingly, we get the following result for the corresponding enumeration problems.

Theorem 5.27.

- ENUM-CLOSEST-STRING$^{\text{LEX}} \in$ DelayFPT *with polynomial space.*
- ENUM-CLOSEST-STRING$^{\text{SIZE}} \in$ DelayFPT *with exponential space.*

Proof. Ω is just the set of operations which flip the i-th bit for every $i \in \mathbb{N}$. Then use Proposition 5.26 and Corollary 5.24. □

5.2.2 Backdoors

In this section we return to the concept of backdoors (for an introduction see Section 4.2 and 4.2.2 on page 42 and 47). Furthermore, to the already known strong backdoor sets, we will introduce *weak* backdoor sets.

If ϕ is a formula and $V \subseteq \mathbf{Vars}(\phi)$ is a set of variables then $\Theta(V)$ is the set of all assignments $\theta \colon V \to \{0,1\}$.

Definition 5.28 (Weak Backdoor Sets)**.**
Denote with $\mathcal{C}$ some class of CNF-formulas, and ϕ be a propositional CNF formula. A set $V \subseteq \mathbf{Vars}(\phi)$ of variables from ϕ is a weak $\mathcal{C}$-backdoor set of ϕ *if there exists an assignment $\theta \in \Theta(V)$ such that $\phi[\theta] \in \mathcal{C}$ and $\phi[\theta]$ is satisfiable.*

Let us recall the definition of strong backdoor sets, be repeating the definition for $\mathcal{C} = \text{HORN}$.

Definition 4.12 (Strong HORN-Backdoor Sets)**.**
A set V of variables of ϕ, $V \subseteq \mathbf{Vars}(\phi)$, is a strong HORN-backdoor set *of ϕ if for all truth assignments $\tau \colon V \to \{0,1\}$ we have $\phi[\tau] \in$ HORN.*

Generally, we want to explicitly define the parametrised version of both backdoor set variants which have been examined here.

Problem:	Weak/Strong-$\mathcal{C}$-Backdoors
Input:	A formula ϕ in 3CNF, $k \in \mathbb{N}$.
Parameter:	The integer k.
Question:	Exists a weak/strong $\mathcal{C}$-backdoor set of size $\leq k$?

Note that the existence of a weak $\mathcal{C}$-backdoor set can be seen as a modification problem where solutions are sequences of variable assignments. The target property then simply is the type of CNF formulas $\mathcal{C}$.

Definition 5.29 (Base Class,[GS12])**.**
The class $\mathcal{C}$ is a base class *if it can be recognised in* P*, satisfiability of its formulas is in* P*, and the class is closed under isomorphisms w.r.t. variable names. We say that $\mathcal{C}$ is* clause defined *if for every CNF-formula ϕ we have: $\phi \in \mathcal{C}$ if and only if $\{C\} \in \mathcal{C}$ for all clauses C from ϕ.*

Proposition 5.30 ([GS12, Prop. 2])**.**
For every clause-defined base class $\mathcal{C}$, detection of weak $\mathcal{C}$-backdoor sets is in FPT *for input formulas in 3CNF.*

In their proof, they describe how utilising a bounded search tree allows to solve the detection of weak $\mathcal{C}$-backdoors in FPT time. Interestingly to note, this technique

results in obtaining *all minimal solutions* in FPT time. This observation results in the following theorem.

Theorem 5.31.
For every clause-defined base class $\mathcal{C}$ and input formulas in 3CNF

- ENUM-WEAK-$\mathcal{C}$-BACKDOORS$^{\text{LEX}} \in$ DelayFPT *with polynomial space, and*
- ENUM-WEAK-$\mathcal{C}$-BACKDOORS$^{\text{SIZE}} \in$ DelayFPT *with exponential space.*

Proof. The set of operations Ω is then "assignTruthValueToVariable(t, i)" for $t \in \{0, 1\}$ and $i \in \mathbb{N}$. A solution then encodes the chosen backdoor sets together with the required assignment. Then use Proposition 5.26 and Corollary 5.24. □

Definition 5.32 ([NRS07; Sze09])**.**
*Let $\mathcal{C}$ be a class of CNF-formulas and ϕ be a CNF-formula. A set $V \subseteq$ **Vars**(ϕ) of variables of ϕ is a* $\mathcal{C}$-deletion backdoor set *of ϕ if $\phi[V]$ is in $\mathcal{C}$, where $\phi[V]$ denotes the formula obtained from ϕ by deleting in ϕ all occurrences of variables from V.*

In the following result we will examine the parametrised enumeration complexity of the task to enumerate all strong $\mathcal{C}$-backdoor sets of a given 3CNF formula for some clause-defined base class $\mathcal{C}$. Crucially, every strong backdoor set has to contain at least one variable from a clause that is not in C which relates to 'hitting all bad clauses' like in the definition of deletion backdoors (see Def. 5.32).

Crucially, strong backdoor sets resemble *deletion backdoor sets* for this kind of base class. A set of variables $V \subseteq$ **Vars**(ϕ) is a deletion backdoor set if and only if $\phi - V \in \mathcal{C}$ where $\phi - V$ denotes the formula that is obtained by deleting all literals over V from clauses in ϕ.

Theorem 5.33.
For every clause-defined base class $\mathcal{C}$ and input formulas in 3CNF

- ENUM-STRONG-$\mathcal{C}$-BACKDOORS$^{\text{LEX}} \in$ DelayFPT *with polynomial space, and*
- ENUM-STRONG-$\mathcal{C}$-BACKDOORS$^{\text{SIZE}} \in$ DelayFPT *with exponential space.*

Proof. We show that for every clause-defined base class $\mathcal{C}$ and input formulas in 3CNF, the problem MIN-STRONG-$\mathcal{C}$-BACKDOORS is FPT-enumerable. Indeed, we only need to branch on the variables from a clause $C \notin \mathcal{C}$ and remove the corresponding literals over the considered variable from ϕ. The size of the branching tree is at most 3^k. As for base classes the satisfiability test is in P, this yields an FPT-algorithm. The neighbourhood function $\mathcal{N}(x, S)$ for $x = (\phi, k)$ is defined to be the set of the pairwise unions of all minimal strong $\mathcal{C}$-backdoors of $(\phi - (S \cup \{x_i\}), k - |S| - 1)$ together with $S \cup \{x_i\}$ for all variables $x_i \notin S$. If **Vars**$(\phi) = \{x_1, \ldots, x_n\}$, then the operations are $\omega_i \colon \phi \mapsto \phi(0/x_i) \wedge \phi(1/x_i)$. One can see, that applying solutions to an instance increases its size by an exponential factor in the parameter which accordingly leads to FPT. □

5.2.3 Weighted Satisfiability Problems

Finally, we reconsider formulas in the Schaefer framework which have been classified without emphasising any order (see Section 4.2 and Section 4.3). As opposed to the previous section where we examine maximum satisfiability questions, we now focus on the problem MINONES-SAT(Γ) defined below.

Definition 5.34 (Minimality)**.**
Given a propositional formula ϕ and an assignment θ over the variables in ϕ with $\theta \models \phi$, we say that θ is minimal *if there does not exist an assignment $\theta' \subset \theta$ which sets strictly less variables to true than θ and $\theta' \models \phi$. The size $|\theta|$ of θ is the number of variables it sets to true.*

Formally, the problems from above are defined as follows:

Problem:	MIN-MINONES-SAT$^{\text{SIZE}}(\Gamma)$
Input:	(ϕ, k), a propositional Γ-formula ϕ, $k \in \mathbb{N}$.
Parameter:	The integer k.
Output:	Generate all inclusion-minimal satisfying assignments θ of ϕ with $\lvert\theta\rvert \leq k$ by non-decreasing size.

Similarly, the problem ENUM-MINONES-SAT(Γ) asks for all satisfying assignments θ of ϕ with $|\theta| \leq k$. In this context, the operations in Ω are "setVariableToTrue(i)" for $i \in \mathbb{N}$.

Theorem 5.35.
For all constraint languages Γ, we have: MIN-MINONES-SAT$^{\text{SIZE}}(\Gamma)$ *is* FPT*-enumerable and* ENUM-MINONES-SAT$^{\text{SIZE}}(\Gamma) \in$ DelayFPT *with exponential space.*

Proof. For the first claim we can simply compute the minimal assignments by a straight forward branching algorithm: initially, begin with the all-0-assignment, then consider all unsatisfied clauses in turn and flip one of the occurring variables to true. The second claim follows by a direct application of Corollary 5.24. □

In the following last result of this section, we consider the parametrised enumeration problem ENUM-MAXONES-SAT$^{\text{SIZE}}(\Gamma)$ (resp., MIN-MAXONES-SAT$^{\text{SIZE}}(\Gamma)$) asks for the set of all (resp., minimal) satisfying assignments θ of given ϕ with $|\theta| \geq k$ by non-decreasing size. The classification employs a result of Marx [Mar05]. He considers the problem EXACTONES-SAT(Γ), where one asks if for a given Γ-formula ϕ and $k \in \mathbb{N}$ there exists a satisfying assignment θ of ϕ with $|\theta| = k$. The problem is shown to be in FPT if and only if Γ has a property called "weakly separable". Further cases which show when this problem is FPT, or even in P, have been considered by Creignou and Vollmer [CV15] in the context of Boolean clones under the aspect of Post's lattice [Pos41].

Theorem 5.36.
For all constraint languages Γ *we have that* ENUM-MAXONES-SAT$^{\text{SIZE}}(\Gamma) \in$ DelayFPT *with exponential space implies* EXACTONES-SAT$(\Gamma) \in$ FPT.

Proof. If ENUM-MAXONES-SAT$(\Gamma) \in$ DelayFPT then we simply check if the first output solution has weight exactly k. If this is true then we accept otherwise we reject. Consequently, we deduce EXACTONES-SAT$(\Gamma) \in$ FPT. □

Enumeration in Poor Man's Propositional Dependence Logic

6

> *Dependence manifests itself in the presence of multitude. A single event cannot manifest dependence, as it may have occurred as a matter of chance.*
>
> — **Jouko Väänänen**
>
> [Vää07, p. 1]

Dependencies are an omnipresent utensil which express a specific relation between phenomena. Such expressions happen in everyday language:"depending on the weekday I will go to work." Physics use them:"the amplitudes of reflection depend on radiation resistance and polarisation." In databases they occur as *functional dependencies*:"the date functionally depends on the order number." Also medicine uses them to express connections between diseases. For mathematics we know them from algebra to express linear dependencies between vectors. Geneticists determined that the biological sex is depending on the allosomes. In social choice theory, one of the significant conditions to design a social welfare function is the independence of irrelevant alternatives. In game theory, strategies can require moves depending on what the opponent does.

Jouko Väänänen is the founding father of *Dependence Logic (DL)* [Vää07] and introduced the notion of dependence to First-Order (FO) Logic. Later, this formalism has been ported to Modal Logic [Vää08] as well. Formally, the semantics of this logic has to be build on sets of assignments which historically have been named *teams*. The notion of teams can be traced back to its origins in the work of Hodges [Hod97a]. We consider a very basic version of DL, namely, the propositional variant: propositional dependence logic. In the following, we investigate the enumeration complexity only of a fragment of this logic, because satisfiability and model-checking in the full logic is NP-complete [EL12; LV13].

Remarkably, DL and its variants connect a vast range of different disciplines resulting in a tremendously growing community [Abr+13]. Many different extensions of these logics have been related to several other areas of research to model phenomena within these. A very recent overview of the area of Dependence Logics is given by Abramsky et al. [Abr+16]. A historical view on this area is presented in Figure 6.1

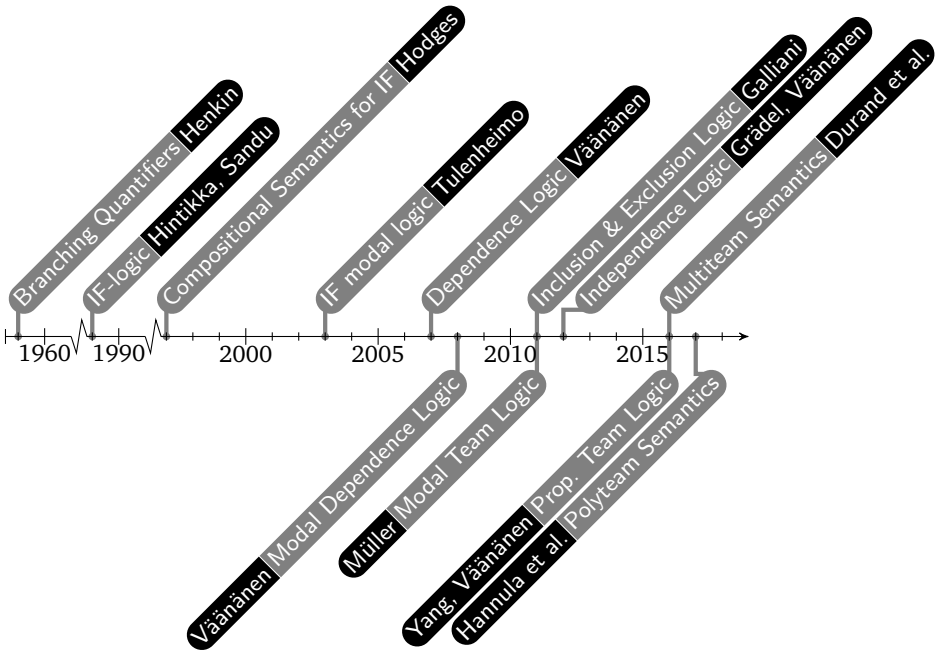


Fig. 6.1: Excerpt from the historical picture of the evolution of Dependence Logic.

6.1 Team-based Propositional Logic

Let $\mathcal{V}$ be a (countably infinite) set of variables. The class of all *Poor Man's Propositional formulas* $\mathcal{PL}^-$ is derived via the grammar

$$\varphi ::= x \mid \neg x \mid 0 \mid 1 \mid \varphi \wedge \varphi,$$

where $x \in \mathcal{V}$. Note that the negation $\neg$ symbol only occurs in front of a variable, that is, we only consider formulas in negation normal form.

Now we will specify the notion of teams and its interpretation on propositional formulas. An *assignment* over $\mathcal{V}$ is a mapping $s\colon \mathcal{V} \to \{0,1\}$. Denote with $\Theta(\mathcal{V}) := \{\, s \mid s \text{ assignment over } \mathcal{V} \,\}$. A *team* T over $\mathcal{V}$ is a subset $T \subseteq \Theta(\mathcal{V})$ of the set of all assignments. Consequently, the set of all teams over $\mathcal{V}$ is denoted by $\mathfrak{P}(\Theta(\mathcal{V}))$ the set of all sets of assignments. If $X \subseteq \mathcal{V}$ is a subset of the variables, we set $T|_X := \{\, s|_X \mid s \in T \,\}$, where $s|_X$ is the restriction of s on X. If T has cardinality $k \in \mathbb{N}$, we say that T is a k-*Team*. If φ is a formula, then a team (assignment) over $\mathbf{Vars}(\varphi)$ is called a *team* (*assignment*) *for* φ.

The set of formulas of *Poor Man's Propositional Dependence Logic* $\mathcal{PDL}^-$ is constructed by the rule set of $\mathcal{PL}^-$ with the extension

$$\varphi ::= =(P, Q),$$

where P, Q are sets of arbitrary variables. We write $=(x_1, x_2, \ldots, x_n)$ as a shorthand for $=(\{x_1, x_2, \ldots, x_{n-1}\}, \{x_n\})$.

Definition 6.1 (Satisfaction)**.**
Let φ be a team-based propositional formula and T be a team for φ. We define $T \models \varphi$ inductively by

$$T \models x \quad :\Leftrightarrow s(x) = 1 \quad \forall s \in T, \qquad T \models \neg x :\Leftrightarrow s(x) = 0 \quad \forall s \in T,$$
$$T \models 1 \quad :\Leftrightarrow \textit{true}, \qquad T \models 0 \quad :\Leftrightarrow T = \emptyset,$$
$$T \models \varphi \wedge \psi \quad :\Leftrightarrow T \models \varphi \textit{ and } T \models \psi,$$
$$T \models =(P, Q) :\Leftrightarrow \forall s, t \in T : s|_P = t|_P \Rightarrow s|_Q = t|_Q.$$

We say that T satisfies φ *if and only if* $T \models \varphi$ *is true.*

Note that we have $T \models (x \wedge \neg x)$ if and only if $T = \emptyset$. This observation motivates the definition for $T \models 0$. Observe that the evaluation in classical propositional logic occurs as the special case of evaluating singletons in team-based propositional logic. Note that many other team-based operators and atoms exist in literature [DKV16].

Definition 6.2 (Downward closure)**.**
A team-based propositional formula φ is called downward closed*, if for every team T we have that $T \models \varphi \Rightarrow \forall S \subseteq T : S \models \varphi$. An operator $\circ$ of arity k is called* downward closed*, if $\circ(\varphi_1, \ldots, \varphi_k)$ is downward closed for all downward closed formulas φ_i, $i = 1, \ldots, k$. A class ϕ of team-based propositional formulas is called* downward closed*, if all formulas in ϕ are downward closed.*

Lemma 6.3.
All atoms and operators in $\mathcal{PDL}^-$ are downward closed. In particular, $\mathcal{PDL}^-$ is downward closed.

Proof. It is easy to see that the atoms x, $\neg x$, 0, 1, $=(\cdot)$ are downward closed. Let φ, ψ be two downward closed formulas and T be a team with $T \models \varphi \wedge \psi$. Then we have $T \models \varphi$ and $T \models \psi$ and by induction hypothesis $S \models \varphi$ and $S \models \psi$ for every subset $S \subseteq T$. It follows that $S \models \varphi \wedge \psi$. For this reason $\wedge$ is downward closed. □

6.2 Group Theory

The following section provides a compact introduction in group actions on sets. For a deeper introduction see, for instance, Rotman's textbook [Rot95] whose notation we will employ. We require these definitions for the enumeration algorithm presented in the following section.

Definition 6.4 (Group action)**.**
Let G be a group with identity element e and X be a set. A group action of G on X*, denoted by $G \circlearrowright X$, is a mapping $G \times X \to X$, $(g, x) \mapsto gx$, with*
(1.) $ex = x \quad \forall x \in X$
(2.) $(gh)x = g(hx) \quad \forall g, h \in G,\ x \in X.$

If G is a group and X is a set then the mapping $(g,h) \mapsto gh$ for $g,h \in G$ defines a group action of G on itself. A group action $G \circlearrowleft X$ induces a group action of G on $\mathfrak{P}(X)$ by $gS := \{ gs \mid s \in S \}$ for all $g \in G,\ S \subseteq X$. Note that this group action preserves the cardinality of sets.

Example 6.5. *Let $G = S_4$ be the group of permutations on the set $S = \{1,2,3,4\}$. Then the group action of G on S can be illustrated as follows (we use cycle notation for permutations):*

$$\begin{aligned}(1\ 2\ 3)(2\ 3\ 4)3 &= 4\\ ((2\ 3)(3\ 4))4 = 2 &= (2\ 3)((3\ 4)4) = (2\ 3)3\\ (1\ 2\ 3\ 4)2 &= 3\end{aligned}$$

Definition 6.6 (Orbit)**.**
Let $G \circlearrowleft X$ be a group action and $x \in X$. Then the orbit of x *is given by*

$$Gx := \{ gx \mid g \in G \} \subseteq X.$$

Proposition 6.7 ([Rot95, p. 56])**.**
Let $G \circlearrowleft X$ be a group action and $x,y \in X$. Then either $Gx = Gy$ or $Gx \cap Gy = \emptyset$. Consequently the orbits of $G \circlearrowleft X$ partition the set X.

Definition 6.8 (Stabiliser)**.**
Let $G \circlearrowleft X$ be a group action and $x \in X$. The stabilizer subgroup of x *is given by $G_x := \{ g \in G \mid gx = x \}$ and indeed is a subgroup of G.*

Proposition 6.9 (Orbit-Stabiliser theorem, [Rot95, Theorem 3.19])**.**
Let G be a finite group acting on a set X. Let $x \in X$. Then the mapping $gG_x \mapsto gx$ is a bijection from G/G_x to Gx. In particular, $|Gx| \cdot |G_x| = |G|$.

The following result has been also discussed in a recent blog post of Lipton and Regan [LR18]. They explain that this result can also be used as a tool to show membership of function problems in #P (sharp-P, or number-P). We will later use it in the context of enumerating teams.

Proposition 6.10 (Cauchy-Frobenius Lemma, [Rot95, Theorem 3.22])**.**
Let G be a finite group acting on a set X. Then the amount of orbits is given by $\frac{1}{|G|} \sum_{g \in G} |\{ x \in X \mid gx = x \}|$.

6.3 Enumeration Complexity

In this section, we will examine the parametrised enumeration complexity of outputting all satisfying teams, that is, the complexity of p-EnumTeam. In what follows, we develop two enumeration algorithms for $\mathcal{PDL}^-$, either guaranteeing FPT delay with exponential space or IncFPT delay with polynomial space.

Let Φ be a class of team-based propositional formulas. We are interested in non-empty teams as solutions, we excluded the $\emptyset$ from the set of all solutions. Nevertheless, formally by the *empty team property*, it is always true that $\emptyset \models \varphi$.

Problem:	p-EnumTeam(Φ)
Input:	A team-based propositional formula $\phi \in \Phi$, $k \in \mathbb{N}$.
Parameter:	The integer k.
Output:	All non-empty teams $T \in \mathfrak{P}(\Theta(\mathbf{Vars}(\varphi)))$ with $\lvert T\rvert \leq k$ s.t. $T \models \varphi$.

Later we will see that the order in which the teams are outputted is significant. We consider two natural orders on teams.

Definition 6.11 (Order of cardinality)**.**
Let R, S *be two teams. Then we define a partial order on the set of all teams by* $R \leq_{size} S :\Leftrightarrow |R| < |S|$ *or* $R = S$.

Given a formula φ, we consider a total order $\leq$ on $\Theta(\mathbf{Vars}(\varphi))$ such that comparing two elements is possible in $O(|\mathbf{Vars}(\varphi)|)$ and iterating over the set of all assignments is feasible with delay $O(|\mathbf{Vars}(\varphi)|)$. By interpreting assignments as a binary encoded integer, we obtain an appropriate order on $\Theta(\mathbf{Vars}(\varphi))$ by translating the order on $\mathbb{N}_0$. If necessary, one could demand that adjacent assignments differ in only one place by using the order induced by the Gray code. Now we are able to define the second order.

Definition 6.12 (Lexicographic order)**.**
Let $R = \{r_1, \ldots, r_n\}$, $S = \{s_1, \ldots, s_m\}$ *be two teams such that* $r_1 < \cdots < r_n$ *and* $s_1 < \cdots < s_m$. *Let* i *be the maximum of* $\{j \leq \min(n, m) \mid r_\ell = s_\ell \text{ for all } \ell \in \{1, \ldots, j\}\} \cap \mathbb{N}$. *Then we define the* lexicographic order *as the a partial order on* $\mathfrak{P}(\Theta(\mathbf{Vars}(\varphi)))$ *by*

$$R \leq_{lex} S :\Leftrightarrow \begin{cases} n \leq m, & i = \min(n, m) \\ r_{i+1} < s_{i+1}, & \textit{else}. \end{cases}$$

Note that the lexicographic order is a total order that does not extend the order of cardinality. For example, we have $\{00, 01, 10\} <_{\text{lex}} \{00, 10\}$ when assignments are ordered according to their integer representation.

Let us define the ordered parametrised enumeration problem with respect to $\leq_{\text{size}}$.

Problem:	p-EnumTeamSize(Φ)
Input:	A team-based propositional formula $\phi \in \Phi$, $k \in \mathbb{N}$.
Parameter:	The integer k.
Output:	All non-empty teams $T \in \mathfrak{P}(\Theta(\mathbf{Vars}(\varphi)))$ with $\lvert T\rvert \leq k$ s.t. $T \models \varphi$ ordered by $\leq_{\text{size}}$.

In the following, we will see that the delay of the constructed algorithm is polynomial in the size of input and the maximal size of an outputted team. Consequently, the parameter being the size of the outputted teams perfectly makes sense to achieve an FPT delay (Theorem 6.35). The sorting of produced solutions, however, is an inherent characteristic of our algorithm as satisfying teams of cardinality k are constructed by analysing those of cardinality $k-1$. As a result, we will notice that enumeration in polynomial space for the more general problem, p-EnumTeam($\mathcal{PDL}^-$), is not possible (Corollary 6.38). Despite of that, we will prove that avoiding exponential space is feasible for p-EnumTeamSize($\mathcal{PDL}^-$) while suffering an increasing delay, that is, IncFPT with polynomial space (Theorem 6.43).

At first we need to introduce some notation before we can construct the algorithm. Let $\varphi \in \mathcal{PDL}^-$ be fixed, $k \in \mathbb{N}_0$,

$$\begin{aligned}
n &:= |\mathbf{Vars}(\varphi)|,\\
\mathcal{T}_k &:= \{T \in \mathfrak{P}(\Theta(\mathbf{Vars}(\varphi))) \mid T \models \varphi, |T| = k\},\\
\mathcal{T}_k^0 &:= \{T \in \mathcal{T}_k \mid (\forall x \in \mathbf{Vars}(\varphi) \mapsto 0) \in T\},\\
t_k &:= |\mathcal{T}_k|,\\
t_k^0 &:= |\mathcal{T}_k^0|.
\end{aligned}$$

An assignment $s \in \Theta(\mathbf{Vars}(\varphi))$ is depicted as a sequence of 0 and 1, precisely: $s = s(x_1)s(x_2)\dots s(x_n)$.

Example 6.13. *For $\varphi := {=}(x_1, x_2)$ we have: $n = 2$ and consequently*

$$\begin{aligned}
\mathcal{T}_2 &= \{\{00, 10\}, \{00, 11\}, \{01, 10\}, \{01, 11\}\},\\
\mathcal{T}_2^0 &= \{\{00, 10\}, \{00, 11\}\},\\
\mathcal{T}_3 &= \mathcal{T}_3^0 = \emptyset.
\end{aligned}$$

In essence, team-based propositional formulas of the form

$$\varphi \equiv \left(\bigwedge_{x \in I} x\right) \wedge \left(\bigwedge_{x \in J} \neg x\right) \wedge \left(\bigwedge_{\ell \in L} {=}(P_\ell, Q_\ell)\right)$$

can be simplified w.l.o.g. to

$$\varphi' := \bigwedge_{\ell \in L} {=}(P'_\ell, Q'_\ell) \quad \text{with} \quad P'_\ell := P_\ell \setminus (I \cup J),\ Q'_\ell := Q_\ell \setminus (I \cup J). \tag{$\star$}$$

Then all satisfying teams for φ can be recovered by extending those for φ'.

Example 6.14. *The formula $x_4 \wedge {=}(x_1, x_3) \wedge {=}(\{x_2\}, \{x_3, x_4\})$ may be reduced to ${=}(x_1, x_3) \wedge {=}(x_2, x_3)$. The team $\{000, 010\}$ satisfies the latter formula and is extended to $\{0001, 0101\}$ in order to satisfy the former one.*

6.3.1 The Group Action of Flipping Bits

Flipping a bit at a fixed position in all assignments of a team T is an invariant for $T \models =(P, Q)$ for sets of propositional symbols P and Q.

Example 6.15. *Consider the four possible 2-teams (teams of cardinality 2) which satisfy* $=(x_1, x_2)$*:*

$$\{00, 10\} \qquad \{00, 11\} \qquad \{01, 11\} \qquad \{01, 10\}$$

Observe that flipping the second bit of $\{00, 10\}$ *and* $\{00, 11\}$ *leads to* $\{01, 11\}$ *and* $\{01, 10\}$*. As a result, it suffices to compute the satisfying teams* $\{00, 10\}$ *and* $\{00, 11\}$ *and constructing the other 2-teams by flipping bits.*

The main concept of the constructed algorithm for ensuring FPT-delay is computing a minor set of satisfying k-Teams and constructing the remaining ones by flipping bits. Identifying each assignment s with the vector $(s(x_1), \ldots, s(x_n))$ lets us obtain a bijection of sets

$$\mathbb{F}_2^n \leftrightarrow \Theta(\mathbf{Vars}(\varphi)).$$

Accordingly, we will switch between interpreting an element as an assignment or an $\mathbb{F}_2$-vector as necessary, leading to expressions like $s+t$ for assignments s and t. Those expressions may seem confusing at first, but become obvious when interpreting s and t as vectors. *Vice versa*, $\mathbb{F}_2$-vectors are considered as assignments that may be contained in a team. Whenever we need to utilise both notations, we indicate this by using $s \in \mathbb{F}_2^n \cong \Theta(\mathbf{Vars}(\varphi))$ instead of simply writing $s \in \mathbb{F}_2^n$ or $s \in \Theta(\mathbf{Vars}(\varphi))$.

By the observation after Definition 6.4 the group action of $(\mathbb{F}_2^n, +)$ on itself induces a group action of $\mathbb{F}_2^n$ on $\mathfrak{P}(\mathbb{F}_2^n)$.

Definition 6.16 (Group action of flipping bits)**.**
The group action $\mathbb{F}_2^n \circlearrowright \mathfrak{P}(\Theta(\mathbf{Vars}(\varphi)))$ *is called* group action of flipping bits.

Let e_i be the i-th standard vector of $\mathbb{F}_2^n$. Then the operation of e_i on the power set of assignments for φ, $\mathfrak{P}(\Theta(\mathbf{Vars}(\varphi)))$, corresponds to flipping the value for x_i in each assignment of a team.

Theorem 6.17.
Let $k \in \mathbb{N}$*. The restriction of* $\mathbb{F}_2^n \circlearrowright \mathfrak{P}(\mathbb{F}_2^n)$ *on* $\mathcal{T}_k$ *yields a group action* $\mathbb{F}_2^n \circlearrowright \mathcal{T}_k$*.*

Proof. As the axioms of group actions are still true on a subset of $\mathfrak{P}(\mathbb{F}_2^n)$, it remains to show that $zT \in \mathcal{T}_k$ for all $z \in \mathbb{F}_2^n$ and $T \in \mathcal{T}_k$. Let $z \in \mathbb{F}_2^n$ and $T \in \mathcal{T}_k$. By Definition 6.4 and as group actions preserve cardinality, we have $|zT| = k$. Let $P \subseteq \mathbf{Vars}(\varphi)$ and $s, t \in \Theta(\mathbf{Vars}(\varphi))$. If $s', t' \in \Theta(\mathbf{Vars}(\varphi))$ arise from s, t by flipping the value for a variable x_i, then obviously

$$s|_P = t|_P \iff s'|_P = t'|_P.$$

It follows that

$$T \models =(P,Q) \iff zT \models =(P,Q) \quad \text{for all } P,Q \subseteq \mathbf{Vars}(\varphi).$$

If φ has the form of $(\star)$ then $zT \models \varphi$ as $T \models \varphi$ proving $zT \in \mathcal{T}_k$. □

Lemma 6.18.
Let $T \in \mathcal{T}_k$, $k \in \mathbb{N}$. Then $\mathbb{F}_2^n T \cap \mathcal{T}_k^0 \neq \emptyset$. For this reason $\mathcal{T}_k^0$ contains a representative systems for the orbits of $\mathbb{F}_2^n \circlearrowleft \mathcal{T}_k$.

Proof. Take $s \in T \subseteq \Theta(\mathbf{Vars}(\varphi)) \cong \mathbb{F}_2^n$. Then $sT \in \mathcal{T}_k^0$ as $z + z = \vec{0}$ for all $z \in \mathbb{F}_2^n$. □

By the previous lemma we can compute $\mathcal{T}_k$ from $\mathcal{T}_k^0$ via generating orbits. In the next step, we present and analyse an algorithm for enumerating those orbits. The results culminate in Theorem 6.22.

Definition 6.19.
Let $\vec{0} \neq s = (s_1, \ldots, s_n) \in \mathbb{F}_2^n$ and $\mathcal{B} \subseteq \mathbb{F}_2^n \setminus \{\vec{0}\}$. Then we define

$$\begin{aligned} \operatorname{last}(s) &:= \max\big\{\, i \in \{1,\ldots,n\} \mid s_i = 1 \,\big\}, \\ \operatorname{last}(\mathcal{B}) &:= \{\, \operatorname{last}(s) \mid s \in \mathcal{B} \,\}. \end{aligned}$$

Definition 6.20.
Let $\mathcal{B}$ be a subset of $\mathbb{F}_2^n$. Then the subspace generated by $\mathcal{B}$ is defined by

$$\operatorname{span}(\mathcal{B}) := \big\{\, b_1 + \cdots + b_r \mid r \in \mathbb{N}_0,\ b_i \in \mathcal{B} \text{ for all } i \in \{1,\ldots,r\} \,\big\}.$$

Lemma 6.21.
Let U be a subspace of the $\mathbb{F}_2$-vector space $\mathbb{F}_2^n$. Let $\mathcal{B} \subseteq U \setminus \{\vec{0}\}$ be a maximal subset such that for all $b, b' \in \mathcal{B}$

$$b \neq b' \Rightarrow \operatorname{last}(b) \neq \operatorname{last}(b') \tag{6.1}$$

Then $\mathcal{B}$ is a basis for U.

Proof. We show by induction on $|A|$ that any set $A \subseteq U \setminus \{\vec{0}\}$ satisfying (6.1) is linearly independent.

IB: If $|A| = 1$ then the claim is obvious.

IS: From (6.1) it follows that there exists an element $a_0 \in A$ with $\operatorname{last}(a_0) > \operatorname{last}(a)$ for all $a_0 \neq a \in A$. For the $\operatorname{last}(a_0)$-th component, clearly, the equation

$$a_0 = \sum_{a_0 \neq a \in A} \lambda_a a, \quad \lambda_a \in \mathbb{F}_2$$

has no solution. $A \setminus \{a_0\}$ is linearly independent by induction hypothesis, then it follows that A is linearly independent.

Now assume for contradiction that $\mathcal{B}$ does not generate U. Choose an $s \in U \setminus \operatorname{span}(\mathcal{B})$ with minimal $\operatorname{last}(s)$. As $\mathcal{B}$ is a maximal subset fulfilling (6.1), for a suitable element $b \in \mathcal{B}$ we have $\operatorname{last}(b) = \operatorname{last}(s)$. But this implies $s - b \in U \setminus \operatorname{span}(\mathcal{B})$ with $\operatorname{last}(s - b) < \operatorname{last}(s)$ and contradicts the minimality of s. □

Theorem 6.22.
Let $T \in \mathcal{T}_k$, $k \in \mathbb{N}$. Then $\mathbb{F}_2^n T$ can be enumerated with delay $O(k^3 \cdot n)$.

Proof. Without loss of generality, let $T \in \mathcal{T}_k^0$. Otherwise, consider the team zT with an arbitrary $z \in T$. Notice that T may have a nontrivial stabiliser subgroup allowing for duplicates when simply applying each $z \in \mathbb{F}_2^n$ to T. Despite of that, by Proposition 6.9, we can enumerate the orbit of T without duplicates when applying a representative system for $\mathbb{F}_2^n/(\mathbb{F}_2^n)_T$.

Considering $\mathbb{F}_2^n$ as a vector space over $\mathbb{F}_2$, the subspaces of $\mathbb{F}_2^n$ correspond to the subgroups of $(\mathbb{F}_2^n, +)$. In view of this, any basis for a complement of the stabiliser subgroup $(\mathbb{F}_2^n)_T$ of T in $\mathbb{F}_2^n$ generates a representative system for $\mathbb{F}_2^n/(\mathbb{F}_2^n)_T$.

Select a basis $\mathcal{B}$ of $(\mathbb{F}_2^n)_T$ as in Lemma 6.21 and set

$$\mathcal{C} := \{ e_i \mid i \in \{1, \ldots, n\} \setminus \operatorname{last}(\mathcal{B}) \},$$

where e_i denotes the i-th standard vector of $\mathbb{F}_2^n$. By construction of $\mathcal{C}$, we can arrange the elements of $\mathcal{B} \cup \mathcal{C}$ such that the matrix containing these elements as columns has triangular shape with 1-entries on its diagonal. Accordingly, $\mathcal{B} \cup \mathcal{C}$ is a basis for $\mathbb{F}_2^n$ and then $\mathcal{C}$ is a basis for a complement of $(\mathbb{F}_2^n)_T$. We explain how to construct $\mathcal{B}$. For $s \in \Theta(\mathbf{Vars}(\varphi)) \cong \mathbb{F}_2^n$ we have

$$s \in (\mathbb{F}_2^n)_T \Rightarrow sT = T \Rightarrow s = s + \vec{0} \in T.$$

As a result, we can compute $(\mathbb{F}_2^n)_T$ by checking $sT = T$ for $|T| = k$ elements in $\mathbb{F}_2^n$. In fact, it is enough to check $sT \subseteq T$ as we have $|sT| = |T|$. Then, we derive $\mathcal{B}$ by inserting each element of $(\mathbb{F}_2^n)_T \setminus \{\vec{0}\}$ preserving (6.1) into $\mathcal{B}$. This shows that Algorithm 6.1 outputs $\mathbb{F}_2^n T$ without duplicates. The delay is dominated by the precomputation phase (lines 1 to 8), which is $O(k^3 \cdot n)$. Observe that we sort the k assignments of each team in ascending order before printing them. □

Example 6.23. *Let $n = 3$ and $T = \{000, 100, 010, 110\}$. Note that T satisfies the reduced formula $=(x_1, x_3) \wedge =(x_2, x_3)$ from Example 6.14 on page 78. We compute the orbit $\mathbb{F}_2^3 T$ of T by Algorithm 6.1. We check $sT = T$ for all nonzero assignments s in T:*

$$100 \cdot T = \{100, 000, 110, 010\} = \{000, 100, 010, 110\} = T,$$
$$010 \cdot T = \{010, 110, 000, 100\} = \{000, 100, 010, 110\} = T,$$
$$110 \cdot T = \{110, 010, 100, 000\} = \{000, 100, 010, 110\} = T.$$

Algorithm 6.1: Enumerating orbits

Input: A team T with $\vec{0} \in T$ and $k = |T|$
Output: The orbit $\mathbb{F}_2^n T$ of T where each outputted team is sorted

```
1  B_last ← ∅;                                              /* Assume that B_last is sorted */
2  for 0⃗ ≠ s ∈ T do                                          /* < k iterations */
3      if last(s) ∈ B_last then continue;                    /* O(n) */
4      failed ← false;
5      for t ∈ T do                                          /* ≤ k iterations */
6          if s + t ∉ T then failed ← true;                  /* O(k · n) */
7      if not failed then B_last ← B_last ∪ {last(s)};       /* O(n) */
8  C_last ← {1, ..., n} \ B_last;                            /* O(n) */
9  for s ∈ span({ e_i | i ∈ C_last }) do
10     Compute sT;                                           /* O(k · n) */
11     Sort sT;                                              /* O(k · n log k) */
12     output sT;
```

On that account we obtain

$$\begin{aligned} \mathcal{B}_{\text{last}} &= \{\text{last}(100), \text{last}(010), \text{last}(110)\} = \{1, 2\}, \\ \mathcal{C}_{\text{last}} &= \{3\}, \\ \text{span}(\{\, e_i \mid i \in \mathcal{C}_{\text{last}} \,\}) &= \{000, 001\}. \end{aligned}$$

Then the orbit of T *is given by*

$$000 \cdot T = \{000, 100, 010, 110\} \text{ and } 001 \cdot T = \{001, 101, 011, 111\}.$$

Finally, we link t_k to t_k^0. The larger the quotient t_k/t_k^0 is, the more computation costs are saved by generating orbits instead of computing $\mathcal{T}_k$ directly.

Theorem 6.24.
Let $k \in \mathbb{N}$ *with* $t_k \neq 0$. *Then we have that* $t_k/t_k^0 = 2^n/k$.

Proof. Lemma 6.18 and $t_k \neq 0$ imply $t_k^0 \neq 0$. As a result, we can choose $T \in \mathcal{T}_k^0$. Now we claim

$$|\mathbb{F}_2^n T \cap \mathcal{T}_k^0| = \frac{k}{|(\mathbb{F}_2^n)_T|}. \tag{6.2}$$

For any $s \in \Theta(\mathbf{Vars}(\varphi)) \cong \mathbb{F}_2^n$ we have that

$$\begin{aligned} sT \in \mathcal{T}_k^0 &\iff \exists t \in T \text{ such that } s + t = \vec{0} \\ &\iff \exists t \in T \text{ such that } s = t \iff s \in T. \end{aligned} \tag{6.3}$$

Consequently, we have $\mathbb{F}_2^n T \cap \mathcal{T}_k^0 = \{\, sT \mid\in T \,\} =: TT$. Let $r, s \in T$ be two elements in T. Then both yield the same team $rT = sT$ if and only if $s \in r(\mathbb{F}_2^n)_T$. Accordingly, for any fixed $r \in T$, we find exactly $|r(\mathbb{F}_2^n)_T| = |(\mathbb{F}_2^n)_T|$ ways of expressing rT in the form of sT, where $s \in T$ by (6.3). While iterating over the k elements sT, $s \in T$, each

team in $\mathcal{T}$ is counted $|(\mathbb{F}_2^n)_T|$ many times. As a result $|\mathcal{T}| = \frac{k}{|(\mathbb{F}_2^n)_T|}$, which in turn proves (6.2).

By Lemma 6.18, there is a representative system $R \subseteq \mathcal{T}_k^0$ for the orbits of $\mathbb{F}_2^n \circlearrowleft \mathcal{T}_k$. Equation (6.2) and the Orbit-Stabiliser Theorem (see Prop. 6.9) implicate

$$\begin{aligned}
t_k &= \sum_{T \in R} |\mathbb{F}_2^n T| && \text{(by Proposition 6.7)} \\
&= \sum_{T \in \mathcal{T}_k^0} \frac{|\mathbb{F}_2^n T|}{|\mathbb{F}_2^n T \cap \mathcal{T}_k^0|} \\
&= \sum_{T \in \mathcal{T}_k^0} \frac{|(\mathbb{F}_2^n)_T|}{k} \cdot |\mathbb{F}_2^n T| && \text{(by (6.2))} \\
&= \sum_{T \in \mathcal{T}_k^0} \frac{|(\mathbb{F}_2^n)_T|}{k} \cdot \frac{2^n}{|(\mathbb{F}_2^n)_T|} && \text{(by Proposition 6.9)} \\
&= \frac{2^n}{k} \sum_{T \in \mathcal{T}_k^0} 1 \\
&= \frac{2^n}{k} t_k^0.
\end{aligned}$$

□

Example 6.25. *Consider again the reduced formula* $\varphi := {=}(x_1, x_3) \wedge {=}(x_2, x_3)$ *from Example 6.14 on page 78. Then the orbits of* $\mathcal{T}_k$*, for* $k \in \mathbb{N}$*, and their corresponding stabiliser subgroups are given in Figure 6.2. Teams located in* $\mathcal{T}_k^0$ *are coloured red. Observe that the number of red teams in each orbit of* $\mathcal{T}_k$ *matches* k *divided by the cardinality of the stabiliser subgroup:*

$$\frac{t_1}{t_1^0} = \frac{8}{1} = \frac{2^3}{1}, \qquad \frac{t_2}{t_2^0} = \frac{16}{4} = \frac{2^3}{2}, \qquad \frac{t_3}{t_3^0} = \frac{8}{3} = \frac{2^3}{3}, \qquad \frac{t_4}{t_4^0} = \frac{2}{1} = \frac{2^3}{4}.$$

We have seen how to construct all satisfying k-teams from a representative system. Subsequently, we need to construct $\mathcal{T}_k^0$. Concerning this, we need to introduce the concept and two results of coherence first.

Definition 6.26 ([Kon13, Def. 3.1])**.**
Let ϕ *be a team-based propositional formula. Then* ϕ *is* k-coherent *if and only if for all teams* T *we have that*

$$T \models \phi \iff \forall R \subseteq T \text{ with } |R| = k \text{ we have } R \models \phi$$

Proposition 6.27 ([Kon13, Prop. 3.3])**.**
The atom ${=}(\cdot)$ *is 2-coherent.*

Proposition 6.28 ([Kon13, Prop. 3.4])**.**
If ϕ*,* ψ *are* k*-coherent then* $\phi \wedge \psi$ *is* k*-coherent.*

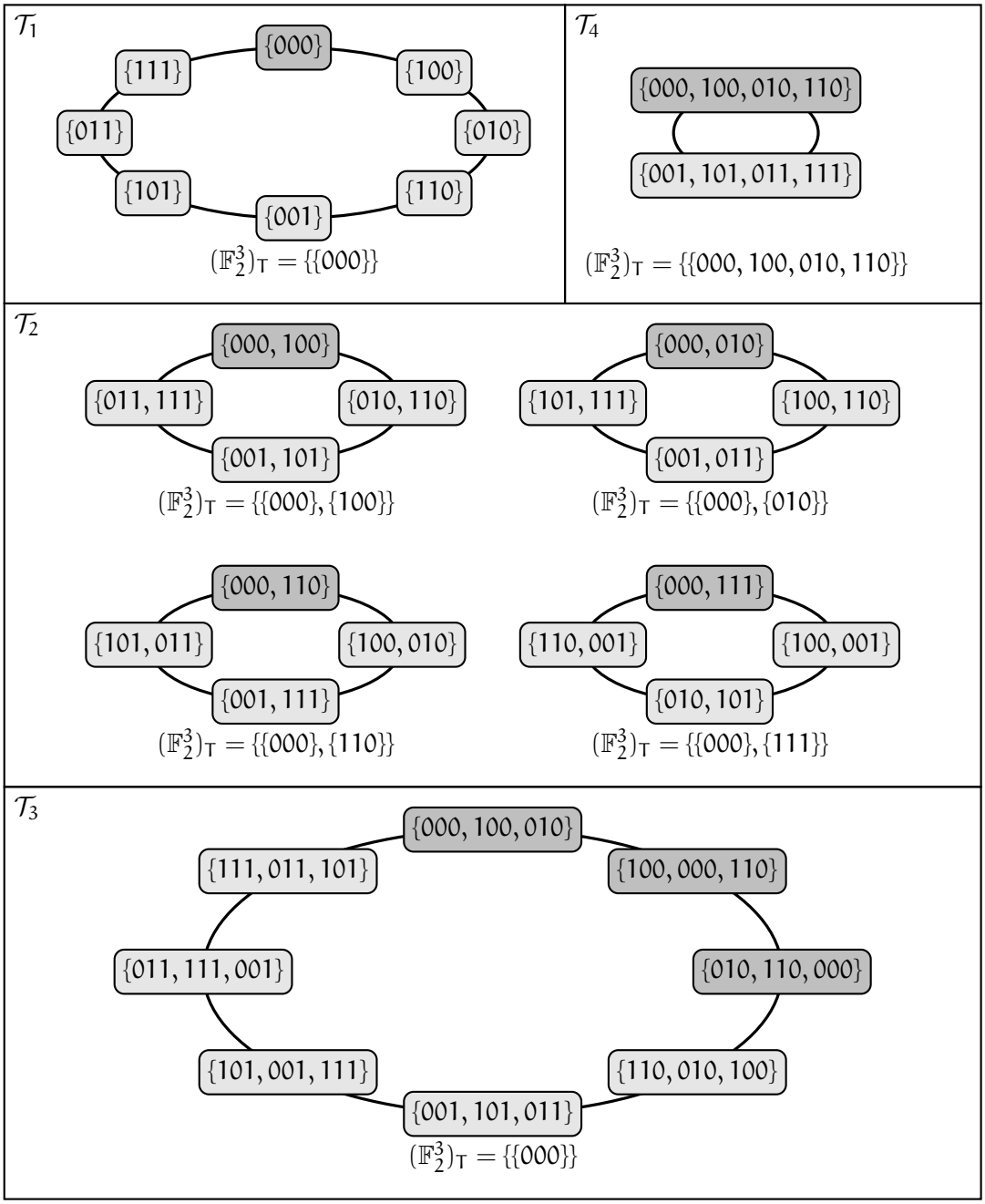


Fig. 6.2: Orbits of $\mathcal{T}_k$ with $\varphi := {=}(x_1, x_3) \wedge {=}(x_2, x_3)$.

Let $T = \{s_1, \dots, s_k\}$ be a team with $s_1 < \cdots < s_k$, $k \geq 2$. Then write

$$T^1_{\mathrm{red}} := \{s_1, \dots, s_{k-1}\},\ T^2_{\mathrm{red}} := \{s_1, \dots, s_{k-2}, s_k\},\ \max(T) := s_k.$$

Later we will see that the following lemma is a powerful tool for constructing the sets $\mathcal{T}^0_k$. In the following proofs we will use the notation T_{red} for some team T as the set of the teams located in $\mathcal{T}^0_k$.

Lemma 6.29.
Let T be as above and $k := |T| \geq 3$. Then the following are equivalent:
(1.) $T \in \mathcal{T}^0_k$,
(2.) $T^1_{\mathrm{red}}, T^2_{\mathrm{red}} \in \mathcal{T}^0_{k-1}$ *and* $\{\vec{0}, s_{k-1} + s_k\} \in \mathcal{T}^0_2$.

Proof. After simplifying φ we may assume that φ is a conjunction of dependence atoms. In particular, φ is then 2-coherent by Proposition 6.27 and 6.28.

(1.) ⇒ (2.): Start with $T \in \mathcal{T}_k$. Note that any subset of cardinality 2 contained in T^1_{red} or T^2_{red} is a subset of T. The 2-coherence of φ yields $T^i_{\mathrm{red}} \models \varphi$ for $i \in \{1, 2\}$. Furthermore, $\vec{0} = s_1 \in T^i_{\mathrm{red}}$ and $|T^i_{\mathrm{red}}| = k-1$ are true. This implicates $T^1_{\mathrm{red}}, T^2_{\mathrm{red}} \in \mathcal{T}^0_{k-1}$. Again, by the 2-coherence of φ, we deduce that $\{s_{k-1}, s_k\} \in \mathcal{T}_2$. Applying the group action $\mathbb{F}^n_2 \circlearrowright \mathcal{T}_2$ shows that $\{\vec{0}, s_{k-1} + s_k\} \in \mathcal{T}^0_2$.

(2.) ⇒ (1.): First, note that $\vec{0} \in T^1_{\mathrm{red}} \subset T$ and $|T| = |T^1_{\mathrm{red}}| + 1 = k$. Now wrongly suppose $T \not\models \varphi$. Then, by 2-coherence, there exists a subset $R \subseteq T$ with $|R| = 2$ and $R \not\models \varphi$. In particular, $R \not\subseteq T^1_{\mathrm{red}}, T^2_{\mathrm{red}}$ is true implying $R = \{s_{k-1}, s_k\}$. This contradicts $s_{k-1}R = \{\vec{0}, s_{k-1} + s_k\} \in \mathcal{T}^0_2$.

□

Now, Algorithm 6.2 computes the sets $\mathcal{T}^0_k$ by utilising Lemma 6.29. To ensure fast list operations, we sort teams by lexicographic order as given in Definition 6.12. When the assignments of each team are saved in ascending order—which is easy to guarantee—the cost of comparing two k-teams is $O(k \cdot n)$. We do not store duplicates, restricting the size of lists containing teams to

$$|\mathfrak{P}(\Theta(\mathbf{Vars}(\varphi)))| = \binom{2^n}{k} \leq (2^n)^k = 2^{k \cdot n}. \tag{6.4}$$

Managing those lists in tries [Knu98, chapter 6.3]. Since any team of cardinality k may be described by kn bits, the standard list operations as searching, insertion and deletion are realised in $O(kn)$. Organise in a way such that all teams of cardinality k differing in their maximal assignment are described by a list $\mathcal{D}_k[T']$, where T' is the team containing the common $k-1$ smaller assignments. It suffices storing the maximal assignment of each T described in $\mathcal{D}_k[T']$ since T is recoverable by T' and $\max(T)$. Accordingly, $\mathcal{D}_k$ becomes a collection of lists indexed by teams of cardinality $k-1$.

Algorithm 6.2: Iteratively constructing $\mathcal{T}^0_k$

Input: $k \in \mathbb{N}$, $k \geq 2$
Dependencies: If $k > 2$: $\mathcal{D}_2[\{\vec{0}\}]$, $\mathcal{D}_{k-1}$ of the previous iteration
Result: $\mathcal{T}^0_k$

```
1  T^0_k ← ∅ D_k ← new Map(Team, List(Assignment));
2  if k = 2 then
3      D_2[{0⃗}] ← ∅;
4      for 0⃗ ≠ s ∈ Θ(Vars(φ)) do                          /* ≤ 2^n iterations */
5          if {0⃗, s} ⊨ φ then                              /* O(|φ|) */
6              D_2[{0⃗}] ← D_2[{0⃗}] ∪ {s};                  /* O(n) */
7              T^0_2 ← T^0_2 ∪ {0⃗, s};                     /* O(n) */
8  else
9      for (T, L) ∈ D_{k-1} do
10         for r ∈ L do                                    /* t^0_{k-1} iterations */
11             T' ← T ∪ {r}, D_k[T'] ← ∅;
12             for s ∈ L with s > r do                     /* ≤ 2^n iterations */
13                 if r + s ∈ D_2[{0⃗}] then                 /* O(n) */
14                     D_k[T'] ← D_k[T'] ∪ {s};             /* O(k · n) */
15                     T^0_k ← T^0_k ∪ {T' ∪ {s}};          /* O(k · n) */
```

With these considerations in mind, we will prove the correctness (Lemma 6.30 and Corollary 6.31) and performance (Corollary 6.32) of the algorithm now.

Lemma 6.30.
Let $k \geq 2$. *For* $T \in \mathcal{T}^0_k$ *we have that* $\max(T) \in \mathcal{D}_k[T^1_{\text{red}}]$. Vice versa, *if* $s \in \mathcal{D}_k[T]$, *then it follows that* $T \cup \{s\} \in \mathcal{T}^0_k$ *and* $s > \max(T)$.

Proof. We prove the result via an induction over k.

IB: ($k = 2$), let $T \in \mathcal{T}^0_2$. It follows that $T^1_{\text{red}} = \{\vec{0}\}$. As we have $\{\vec{0}, \max(T)\} \models \varphi$, in line 6 $\max(T)$ is inserted into $\mathcal{D}_2[T^1_{\text{red}}]$. Now let $s \in \Theta(\mathbf{Vars}(\varphi))$ and $T \in \mathfrak{P}(\Theta(\mathbf{Vars}(\varphi)))$ such that $s \in \mathcal{D}_2[T]$. The only team occurring in $\mathcal{D}_2$ is $T = \{\vec{0}\}$. We have $s \in \mathcal{D}_2[T]$ if and only if $T \cup \{s\} = \{\vec{0}, s\} \models \varphi$ and $s \neq \vec{0}$. The claim follows.

IS: ($k - 1 \to k$), let $T = \{s_1, \ldots, s_k\} \in \mathcal{T}^0_k$, $s_1 < \cdots < s_k$. By induction hypothesis and Lemma 6.29 we have that $s_{k-1} \in \mathcal{D}_{k-1}[T^1_{\text{red}} \setminus \{s_{k-1}\}]$. Accordingly, the loop body of line 10 is invoked with $T \triangleq T^1_{\text{red}} \setminus \{s_{k-1}\}$, $r \triangleq s_{k-1}$, $T' \triangleq T^1_{\text{red}}$. Furthermore by Lemma 6.29 the loop body of line 12 is invoked with $s \triangleq s_k$, passing the check in line 13. As a result, $s_k = \max(T)$ is inserted into $\mathcal{D}_k[T^1_{\text{red}}]$.

Now let $s \in \Theta(\mathbf{Vars}(\varphi))$ and $T \in \mathfrak{P}(\Theta(\mathbf{Vars}(\varphi)))$ such that $s \in \mathcal{D}_k[T]$. Then by the construction of $\mathcal{D}_k$ there exist a team T' and $r \in \mathcal{D}_{k-1}[T']$ with $r < s$, $s \in \mathcal{D}_{k-1}[T']$, $T' \cup \{r\} = T$ and $\{\vec{0}, r+s\} \in \mathcal{T}^0_2$. The induction hypothesis yields $T' \cup \{r\}, T' \cup \{s\} \in \mathcal{T}^0_{k-1}$ and $r > \max(T')$, implying $s > r = \max(T)$. As s and r are the largest elements of

$T \cup \{s\}$, it follows that $(T \cup \{s\})^1_{\mathrm{red}} = T' \cup \{r\}$ and $(T \cup \{s\})^2_{\mathrm{red}} = T' \cup \{s\}$. By Lemma 6.29 we obtain $T \cup \{s\} \in \mathcal{T}^0_k$. □

Corollary 6.31.
Algorithm 6.2 constructs the sets $\mathcal{T}^0_k$ correctly.

Proof. Every team T inserted into $\mathcal{T}^0_k$ by Algorithm 6.2 is of the form $T' \cup \{s\}$ with $s \in \mathcal{D}_k[T']$. Then by Lemma 6.30 it follows that $s > \max(T)$ and $T \in \mathcal{T}^0_k$. Observe that the decomposition of T is unique due to $s > \max(T)$. As a result, T is inserted only once and duplicates cannot occur.

Now consider the case $T \in \mathcal{T}^0_k$. Lemma 6.30 claims $\max(T) \in \mathcal{D}_k[T^1_{\mathrm{red}}]$. After adding $\max(T)$ into $\mathcal{D}_k[T^1_{\mathrm{red}}]$, Algorithm 6.2 inserts $T^1_{\mathrm{red}} \cup \max(T) = T$ into $\mathcal{T}^0_k$. □

Corollary 6.32.
Algorithm 6.2 requires $t^0_{k-1} \cdot 2^n \cdot O(k|\varphi|)$ time on input $k \in \mathbb{N}$.

Proof. Note that for all $\vec{0} \neq s \in \Theta(\mathbf{Vars}(\varphi))$ we have

$$\{\vec{0}, s\} \models {=}(P, Q) \iff \{s\} \models \left(\bigvee_{x \in P} x\right) \vee \left(\bigwedge_{y \in Q} \neg y\right).$$

As a consequence, $\{\vec{0}, s\} \models \varphi$ can be accomplished in linear time by evaluating a $\mathcal{PL}$-formula of length $O(|\varphi|)$ (where $\vee$ has the classical propositional disjunction semantics). Accessing the list $\mathcal{D}_k[T]$ for a team T is in $O(k \cdot n)$ if $\mathcal{D}_k$ is implemented as a trie. If $\mathcal{D}_k[T]$ is realised in the same way then its operations are in $O(\log 2^n) = O(n) \subseteq O(k \cdot n)$.

As the decomposition for $T \in \mathcal{T}^0_{k-1}$ into T' and s with $s > \max(T')$ is unique, applying Lemma 6.30 yields that the loop body of line 10 is invoked t^0_{k-1} times.

Taking into account that $n \leq |\varphi|$, we obtain the claim by adding up all costs. □

Example 6.33. *We construct the sets $\mathcal{T}^0_k$ for the reduced formula $\varphi := {=}(x_1, x_3) \wedge {=}(x_2, x_3)$ from Example 6.14. Trivially, we have that $\mathcal{T}^0_1 = \{\{000\}\}$. When computing $\mathcal{T}^0_2$, we have to identify all nonzero assignments that satisfy $(x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3)$. Obviously, the satisfying assignments are 100, 010, 110 and 111. We obtain $\mathcal{D}_2[\{000\}] = \{100, 010, 110, 111\}$. Figure 6.3 illustrates the construction of the remaining lists and the resulting sets $\mathcal{T}^0_k$. We are able to verify that the orbits presented in Example 6.25 are exactly those of $\mathcal{T}_k$, $k \in \mathbb{N}$. Each orbit contains at least one element of $\mathcal{T}^0_k$ and every team in $\mathcal{T}^0_k$ can be recovered in one orbit of Figure 6.2.*

Although by Corollary 6.32 Algorithm 6.2 does not perform in FPT time on input $k \in \mathbb{N}$, we can ensure FPT delay when distributing its execution over the process of outputting all satisfying teams of cardinality $k - 1$. For this reason we investigate the costs of computing $\mathcal{T}^0_k$ divided by t_{k-1}. With Corollary 6.32 and

$$k - 1 = \frac{t^0_{k-1} \cdot 2^n}{t_{k-1}},$$

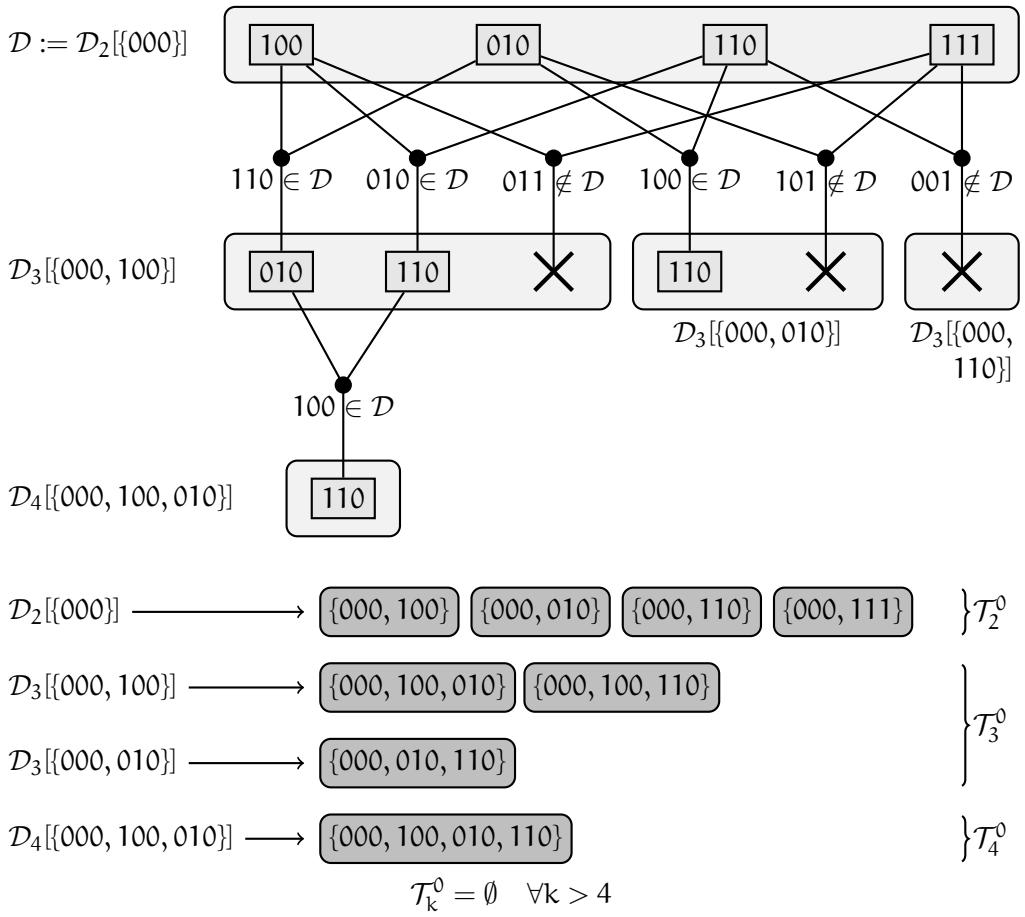


Fig. 6.3: Construction of $\mathcal{T}_k^0$ with $\varphi := =(x_1, x_3) \wedge =(x_2, x_3)$.

Algorithm 6.3: Enumerating satisfying teams in $\mathcal{PDL}^-$, ordered by cardinality

Input: A team-based propositional formula φ as in Equation $(\star)$, $k \in \mathbb{N}$
Output: All teams T for φ with $T \models \varphi$, $1 \leq |T| \leq k$

```
1  T_1^0 ← {{0⃗}};
2  for ℓ = 2, ..., k + 1 do
3      simultaneously
4          while T_{ℓ−1}^0 ≠ ∅ do
5              Choose T ∈ T_{ℓ−1}^0;
6              for T′ ∈ F_2^n T (Algorithm 6.1) do
7                  output T′;
8                  T_{ℓ−1}^0 ← T_{ℓ−1}^0 \ {T′};
9      simultaneously Compute T_ℓ^0 by Algorithm 6.2;
10     if T_ℓ^0 = ∅ then break;
```

which is a transformation of the equation in Theorem 6.24, we obtain

$$\frac{\text{computationCosts}(\mathcal{T}_k^0)}{t_{k-1}} = \frac{t_{k-1}^0 \cdot 2^n \cdot O(k|\varphi|)}{t_{k-1}} = (k-1) \cdot O(k|\varphi|) = O(k^2|\varphi|).$$

Consequently, the delay of Algorithm 6.3 is bounded by $O(k^3|\varphi|)$ (as the delay of generating $\mathbb{F}_2^n T$ is $O(k^3 n)$ by Theorem 6.22). Note that the cost of removing elements in $\mathcal{T}_k^0$, which is $O(k \cdot n)$ is contained in $O(k^3|\varphi|)$. Practically, we need to interleave both computation strands by executing k iterations of the loop at line 12 in Algorithm 6.2 whenever a team is outputted.

Lemma 6.34.
Algorithm 6.3 enumerates all satisfying teams T for φ with $1 \leq |T| \leq k$ without duplicates.

Proof. It is easy to see that all dependencies in Algorithm 6.2 and 6.3 are resolved in time. By Proposition 6.7 and Lemma 6.18 every satisfying team is outputted at least once. By removing every outputted element in line 8 no orbit is outputted twice, preventing duplicates. □

Finally, we can conclude.

Theorem 6.35.
p-EnumTeamSize$(\mathcal{PDL}^-) \in$ DelayFPT *with exponential space.*

6.3.2 Limiting Memory Space

In this subsection, we examine the memory requirements of Algorithm 6.3. Throughout the execution, $\mathcal{D}_2[\{\vec{0}\}]$, $\mathcal{D}_k$ and $\mathcal{T}_k^0$ have to be saved. However, the size of those lists increases exponentially when raising the size of the outputted teams or the amount of variables occurring in the formula φ. By Equation (6.4) from page 85 Algorithm 6.3 requires $O(2^n)$ when fixing the parameter κ. In fact, any algorithm that saves a representative system for the orbits of $\mathbb{F}_2^n \circlearrowleft \mathcal{T}_k$ cannot perform in polynomial

space by the following theorem. For this reason, we have to discard the group action of flipping bits when limiting memory space to polynomial sizes.

Lemma 6.36.
Let $k > 1$ *and* $n \in \mathbb{N}$. *We set* $\varphi := =(x_1, x_2, \ldots, x_n)$. *Then the amount of orbits of* $\mathbb{F}_2^n \circlearrowleft \mathcal{T}_k$ *is not polynomial in* n.

Proof. Note that each orbit of $\mathbb{F}_2^{n-1}$ on the set of all k-teams over $n-1$ variables maps to an orbit of $\mathbb{F}_2^n \circlearrowleft \mathcal{T}_k$ by extending all assignments of a team so that x_n is assigned to the same value. As we have

$$f(n) \in n^{O(1)} \Leftrightarrow f(n-1) \in n^{O(1)}$$

for any function $f\colon \mathbb{N} \to \mathbb{N}$, we may assume that φ is equivalent to 1 with $|\mathbf{Vars}(\varphi)| = n$.

By the Cauchy-Frobenius Lemma (see Proposition 6.10 on page 76) the amount of orbits is at least

$$\frac{|\{T \in \mathfrak{P}(\Theta(\mathbf{Vars}(\varphi))) \mid |T| = k\}|}{2^n}$$

when neglecting all summands except the one for $\vec{0} \in \mathbb{F}_2^n$. That is why the number of orbits in $\mathcal{T}_k$ has to be larger than $\binom{2^n}{k}/2^n$, which already increases exponentially in n. □

In the previous subsection, we indirectly limited the cardinality of outputted teams by considering it as the parameter to show membership in DelayFPT. As the following theorem shows, this measure is also necessary when demanding polynomial space.

Theorem 6.37.
Let Φ *be any fragment of team-based propositional logic s.t. and* κ *be a parametrisation with* $\kappa \notin n^{O(1)}$ *such that for any* $n \in \mathbb{N}$ *there exists a formula* $\varphi_n \in \Phi$ *in* n *variables with at least* $2^{\kappa(n)}$ *satisfying teams. Then it follows that* p-EnumTeam(Φ) *cannot be enumerated with polynomial space.*

Proof. Any enumeration algorithm enumerating $\mathrm{Sol}(\varphi_n)$ has to output $2^{\kappa(n)}$ different teams and the same amount of configurations have to be adopted. To distinguish these, the configurations are encoded by at least $\kappa(n)$ bits. However, considering a RAM performing in polynomial space requires the contents of all registers to be encoded by a polynomial amount of bits. For this reason, a RAM enumerating $\mathrm{Sol}(\varphi_n)$ cannot perform in polynomial space. □

Corollary 6.38.
The problem p-EnumTeam($\mathcal{PDL}^-$) *cannot be enumerated in polynomial space.*

Proof. Use Lemma 6.36 in combination with Theorem 6.37. □

Algorithm 6.4: Enumerating satisfying teams in polynomial space, ordered by cardinality

Input: A team-based propositional formula φ as in Equation $(\star)$
Output: All teams T for φ with $T \models \varphi$, $1 \leq |T| \leq \kappa(\varphi)$

```
1  for k = 1, ..., κ(φ) do
2      T ← {s_first};
3      while true do
4          if |T| = k and T ⊨ φ then output T;
5          s ← max(T);
6          if |T| < k and T ⊨ φ and s ∈ hasNext then  T ← T ∪ {next(s)};
7          else if s ∈ hasNext then  T ← T \ {s} ∪ {next(s)};
8          else if |T| > 1 then
9              T ← T \ {s};
10             s ← max(T);
11             T ← (T \ {s}) ∪ {next(s)};
12         else  break;
```

We now present an algorithm enumerating p-EnumTeamSize($\mathcal{PDL}^-$). Compared to Algorithm 6.3, it saves memory space by recomputing the satisfying teams of lower cardinality instead of storing them in a list. As a downside we have to allow incremental delays.

Let us define a unary relation `hasNext` on $\Theta(\mathbf{Vars}(\varphi))$ by $s \in \texttt{hasNext}$ if and only if $\exists t \in \Theta(\mathbf{Vars}(\varphi))$ s.t. $s < t$. For any $s \in \texttt{hasNext}$ let $\texttt{next}(s)$ be the unambiguous assignment such that we have $s < \texttt{next}(s)$ but $s < t < \texttt{next}(s)$ is not true for any assignment t. Let s_{first} denote the smallest element in $\Theta(\mathbf{Vars}(\varphi))$ and s_{last} denote the largest one. As already mentioned when defining the lexicographical order, we assume that `hasNext`, `next` and s_{first} may be determined in $O(n)$ time.

Lemma 6.39.
Let T be a team with cardinality k. Then $T \models \varphi$ can be checked in $O(k^2|\varphi|)$ time.

Proof. Because of the 2-coherence of φ it is enough to check all 2-subteams of T. By the proof of Corollary 6.32 checking a 2-team requires $O(|\varphi|)$ time. As T has $O(k^2)$ 2-subteams, the claim follows. □

Let $k \in \mathbb{N}$ and write $\mathcal{M}_k$ for the set of teams T that is assigned to during the k-th iteration of the outer loop of Algorithm 6.4.

Lemma 6.40.
Let $S \in \mathcal{M}_k$ be a nonempty set such that $s := \max(S) \in \texttt{hasNext}$. Then it follows that $(S \setminus \{s\}) \cup \{\texttt{next}(s)\} \in \mathcal{M}_k$. In particular, we have $(S \setminus \{s\}) \cup \{t\} \in \mathcal{M}_k$ for all $t \in \Theta(\mathbf{Vars}(\varphi))$ with $t \geq s$.

Proof. We prove the result by an induction over $k - |S|$.

IB: ($|S| = k$), as we have $|S| \not< k$ and $s \in \texttt{hasNext}$, line 7 is executed and T is assigned to $(S \setminus \{s\}) \cup \{\texttt{next}(s)\}$. ✓

IS: ($|S|+1 \to |S|$, $1 \leq |S| < k$), if $S \not\models \varphi$, line 7 is executed and the claim follows. If $S \models \varphi$, line 6 follows and then $S \cup \{\texttt{next}(s)\} \in \mathcal{M}_k$. By induction hypothesis, $S \cup \{s_{\text{last}}\} \in \mathcal{M}_k$. When executing the body of the while loop with T assigned to $S \cup \{s_{\text{last}}\}$, the block beginning at line 9 is executed, assigning T to $(S \setminus \{s\}) \cup \{\texttt{next}(s)\}$.

□

Lemma 6.41.
Let $k \in \mathbb{N}$ and S be a team with $S \models \varphi$ and $|S| \leq k$. Then it follows that $S \in \mathcal{M}_k$.

Proof. We prove the result by an induction over $|S|$.

IB: ($|S| = 1$), clearly $\{s_{\text{first}}\} \in \mathcal{M}_k$. By Lemma 6.40 every 1-team is contained in $\mathcal{M}_k$.

IS: ($|S| - 1 \to |S|$, $1 < |S| \leq k$), start with $s = \max(S)$. Since φ is downward closed, we have that $S \setminus \{s\} \models \varphi$. The induction hypothesis then yields $S \setminus \{s\} \in \mathcal{M}_k$. Consequently, the while loop is executed with T assigned to $S \setminus \{s\}$. Then, line 7 is executed, assigning T to a team $(S \setminus \{s\}) \cup \{t\}$, where t is an appropriate assignment with $t \leq s$. Finally, Lemma 6.40 yields $S \in \mathcal{M}_k$.

□

Lemma 6.42.
There exists a polynomial p and a recursive function f such that the i-th delay of Algorithm 6.4 is bounded by $f(\kappa(\varphi)) \cdot i^2 p(|\varphi|)$.

Proof. Note that the delay is constant when outputting the 2^n singletons that satisfy φ trivially. As a result, we assume that $i \geq 2^n$. It is easy to verify that any team T is assigned to is lexicographically larger than the previous value for T. For this reason the number of iterations of the inner while loop is bounded by $|\mathcal{M}_k|$.

As T is not assigned to teams with greater cardinality when the current value for T does not satisfy φ, it follows that $S \setminus \{\max(S)\} \models \varphi$ for any $S \in \mathcal{M}_k$ with $|S| > 1$. Consequently, it follows

$$|\mathcal{M}_k| \leq 2^n \sum_{l=0}^{k-1} t_l \leq i \sum_{l=0}^{k-1} t_l.$$

Let S be the $(i+1)$-th outputted element and set $k = |S|$. As $S \in \mathcal{M}_k$ and $|S| > 1$, by outputting teams of lower cardinality first we guarantee that $i \geq \sum_{l=1}^{k-1} t_l$. Furthermore, S is outputted in the k-th iteration of the outer loop. Consequently, the inner while loop has been executed at most $k \cdot i^2$ times before outputting S. Since k is bounded by a $\kappa(\varphi)$, by Lemma 6.39 it follows that the body of the inner while loop can be executed in FPT time. As one loop iteration adds a faktor of $\kappa(\varphi)^2 \cdot |\varphi|)$ (see Lemma 6.39), we conclude that the i-th delay is bounded by $\kappa(\varphi)^3 \cdot i^2 \cdot p(|\varphi|)$, where p is an appropriate polynomial.

Now let i be the total amount of outputted teams. Then the number of iterations of the inner while loop is bounded by

$$\sum_{k=1}^{\kappa(\varphi)} |\mathcal{M}_k| \leq \kappa(\varphi) \cdot |\mathcal{M}_{\kappa(\varphi)}| \leq \kappa(\varphi) \cdot 2^n \sum_{\ell=0}^{\kappa(\varphi)} t_\ell \leq \kappa(\varphi) \cdot i^2.$$

Accordingly, we have an IncFPT-delay. □

Theorem 6.43.
Algorithm 6.4 is an IncFPT*-algorithm for* p-EnumTeamSize($\mathcal{PDL}^-$) *which performs in polynomial space.*

Proof. The algorithm saves only one team of cardinality $\leq \kappa(\varphi)$ and one assignment for which $(\kappa(\varphi) + 1)$ registers are required. By Lemma 6.41 it is clear that the algorithm outputs the satisfying teams ordered by cardinality. Lemma 6.42 states that the delays conform to the definition of IncFPT. □

Page left intentionally blank to have matching page numbers with the printed version.

Conclusion

7

> *Nihil est sine ratione.*
> *There is nothing without a reason.*
>
> **— Gottfried Wilhelm Leibniz**

In this thesis, we made the first step of developing a computational complexity theory for parametrised enumeration problems. It has been shown that the notion of the parametrised enumeration complexity class DelayFPT is useful to classify several problems with similar enumeration procedures. Similarly as in the classical setting, we show that IncFPT relates to DelayFPT as IncP relates to DelayP. We further underlined this observation later in Chapter 5 while studying trade-offs between delay and space. In Section 3.2 we proved the first connections to classical enumeration complexity by showing that a collapse of OutputFPT to IncFPT implies collapsing OutputP to CapIncP and *vice versa*. While proving this result, we showed equivalences of collapses of parametrised function classes developed in this paper to collapses of classical function classes. In particular, we proved that $\mathsf{TF}(\mathsf{para\text{-}NP}) = \mathsf{F}(\mathsf{FPT})$ if and only if $\mathsf{TF}(\mathsf{NP}) = \mathsf{FP}$. The function complexity class $\mathsf{TF}(\mathsf{NP})$, which has $\mathsf{TF}(\mathsf{para\text{-}NP})$ as its parametrised counterpart, contains significant cryptography-related problems such as factoring. Furthermore, we studied a parametrised incremental FPT time enumeration hierarchy on the level of exponent slices (Def. 3.3) and observed that $\mathsf{CapIncFPT}_1 = \mathsf{DelayFPT}$. Also, an interleaving of the two hierarchies, IncFPT_α and $\mathsf{CapIncFPT}_\alpha$, has been shown. These results underline that parametrised enumeration complexity is an area worthwhile to study as there are deep connections to the classical field.

Then, Chapter 4 translated the techniques of kernelisation and self-reducibility to our new framework. In particular, we defined an enum-kernelisation as a preprocessing step suitable for efficient enumeration and even characterised with it the class DelayFPT. Subsequently, we exemplified this technique at the problem Enum-Vertex-Cover. Furthermore, we observed that every problem in DelayFPT can be enumerated with an FPT preprocessing time followed by only (classical) polynomial delay proving how effective a single FPT precomputation phase is. Subsequently, we used self-reducibility for satisfiability related problems to obtain efficient parametrised enumeration algorithms for exact strong backdoor sets and maximum weighted satisfiability. While it is clear that obtaining such an efficient enumeration requires the corresponding decision problem to be in FPT, we proved that for Enum-MaxOnes-SAT this is not enough. More precisely, we showed that Enum-MaxOnes-SAT obeys a dichotomy which, surprisingly, is different from its decision variant MaxOnes-SAT.

In the second part of this thesis, we extended our framework to comply with mathematical orders on the solution space, and demonstrate the meaningfulness of our framework for a large variety of problems: cluster editing, chordal completion, closest-string, weak and strong backdoors, minimum and maximum satisfiability. An important point of our research is proposing a very general strategy for efficient enumeration. This is particularly rather rare in the literature where usually algorithms are devised individually for specific problems. Most notably, our scheme not only yields DelayFPT algorithms for all graph modification problems that are characterised by a finite set of forbidden patterns, but it also provides results for *general* modification problems. Here we validated our definitions by diversely encoded problems: problems over strings, formulas, and constraints.

Significant to note, all results of problems with respect to the order *size* require exponential space due to the inherent use of the priority queues to achieve *ordered* enumeration. An interesting question is whether there exists a method which requires less space but uses a comparable delay between the output of solutions and still obeys the underlying order on solutions.

The third part considered enumerating satisfying teams for a fragment of propositional dependence logic. It turned out that this is a difficult task when sorting the output by its cardinality. We achieved that the problem is in DelayFPT when the parameter is chosen to be the team size. If one focusses the space requirements of this algorithmic task, we established that any algorithm saving a representative system for the orbits of $\mathbb{F}_2^n \circlearrowleft \mathcal{T}_k$ cannot perform in polynomial space. As another negative remark, we want to point out that allowing for split junction (and accordingly talking about full $\mathcal{PDL}$) will not yield any DelayFPT or DelayP algorithms in our setting unless $\mathsf{P} = \mathsf{NP}$.

Lastly, we would like to mention that the algorithms enumerating orbits and the satisfying teams, respectively, can be modified such that satisfying teams for formulas of the form $\varphi_1 \mathbin{⩒} \varphi_2 \mathbin{⩒} \cdots \mathbin{⩒} \varphi_r$ with $r \in \mathbb{N}$, $\varphi_i \in \mathcal{PDL}^-$ can be enumerated, where $⩒$ is the classical disjunction. The idea is to merge the outputs $\mathrm{Sol}(\varphi_i)$, $i \in \{1, \ldots, r\}$, which is possible in polynomial delay if the output for each φ_i is pre-sorted according to a total order.

As a future aspect for further research, it remains open to identify the (parametrised) enumeration complexity of Poor Man's Propositional Dependence Logic when other orders such as the lexicographical order, are considered. Besides, the conjunction free fragment of $\mathcal{PDL}$ permitting the split junction operator remained unclassified.

Finally, all classified problems with the corresponding pointer to the result in this thesis are depicted in Table 7.1.

Problem	Class	Space	Reference
Enum-Krom	DelayP	pol.	Ex. 2.11
Enum-Vertex-Cover	DelayFPT	exp.	Cor. 4.4
Exact-Strong-BackDoorSet[Horn]	DelayFPT	exp.	Thm. 4.13
Enum-$\mathcal{M}_{\mathcal{P}}^{\text{Lex}}$	DelayFPT$^{\heartsuit}$	pol.	Thm. 5.9
Enum-Triangulation$^{\text{Lex}}$	DelayFPT	pol.	Cor. 5.11
Enum-$\mathcal{M}_{\mathcal{P}}^{\text{Size}}$	DelayFPT$^{\clubsuit}$	exp.	Thm. 5.13
Enum-Triangulation$^{\text{Size}}$	DelayFPT	exp.	Cor. 5.15
Enum-Cluster-Editing$^{\text{Size}}$	DelayFPT	exp.	Cor. 5.17
Enum-Triangle-Deletion$^{\text{Size}}$	DelayFPT	exp.	Cor. 5.17
Enum-Closest-String$^{\text{Lex}}$	DelayFPT	pol.	Thm. 5.27
Enum-Closest-String$^{\text{Size}}$	DelayFPT	exp.	Thm. 5.27
Enum-Weak-$\mathcal{C}$-Backdoors$^{\text{Lex}}$	DelayFPT†	pol.	Thm. 5.31
Enum-Weak-$\mathcal{C}$-Backdoors$^{\text{Size}}$	DelayFPT†	exp.	Thm. 5.31
Enum-Strong-$\mathcal{C}$-Backdoors$^{\text{Lex}}$	DelayFPT†	pol.	Thm. 5.33
Enum-Strong-$\mathcal{C}$-Backdoors$^{\text{Size}}$	DelayFPT†	pol.	Thm. 5.33
Min-MinOnes-SAT$^{\text{Size}}(\Gamma)$	FPT-enum.	exp.	Thm. 5.35
Enum-MinOnes-SAT$(\Gamma)^{\text{Size}}$	DelayFPT	exp.	Thm. 5.35
p-EnumTeamSize$(\mathcal{PDL}^{-})$	DelayFPT	exp.	Thm. 6.35
p-EnumTeamSize$(\mathcal{PDL}^{-})$	IncFPT	pol.	Thm. 6.43

Tab. 7.1: Overview of considered problems in chronological order.
$^{\heartsuit}$: if $\mathcal{M}_{\mathcal{P}} \in$ FPT.
$^{\clubsuit}$: if $\mathcal{N}_{\mathcal{M}_{\mathcal{P}}} \in$ FPT.
†: for clause-defined base class $\mathcal{C}$.

Page left intentionally blank to have matching page numbers with the printed version.

Outlook

8

> *We can only see a short distance ahead, but we can see plenty there that needs to be done.*
>
> **— Alan Turing**

Very recently, Creignou et al. proposed new enumeration complexity classes for hard problems [Cre+17a]. Intuitively, these classes embody the idea of the polynomial hierarchy on the enumeration complexity level. Formally, the authors present oracle classes of the form $\mathsf{DelayP}^{\mathcal{C}}$ by RAMs which have access to oracles from the class $\mathcal{C}$ but still obey an overall polynomial delay. Additionally, a subtle but meaningful restriction is made by introducing the class $\mathsf{DelayP}^{\mathcal{C}}_{\mathrm{p}}$, that is, the input size of the oracle calls have to be bounded by a polynomial in the original input size. Besides, the authors discuss relations between variants of incremental time and polynomial delay. It would be engaging to study this approach in the framework of parametrised enumeration problems. Are there problems which can be pinned to a class as, for instance, "DelayW[1]"? Or what insights might bring studying classes of the form $\mathsf{DelayFPT}^{\mathsf{W[1]}}$? Furthermore, Creignou et al. provide tools for reductions in this context. Such a technique for translating a parametrised enumeration problem into another is still missing and would improve understanding the interplay of parametrised enumeration problems significantly.

As we explained in Chapter 3, currently, the understanding of the class IncFPT is still in its infancy. In Chapter 6, we made some fascinating results on trading space with time: We avoided exponential space (and reached polynomial space) while paying with IncFPT instead of DelayFPT as before. At the current stage, it is not clear if this is a general phenomenon, that is, every problem in DelayFPT with exponential space can be solved in IncFPT with polynomial space. One puzzle piece might be constructing solution search trees on the fly (see Theorems 6.35 and 6.43), though we cannot say if this suffices alone. It is worth to mention that the analogue of this question on the classical enumeration side, that is, understanding the relation between IncP and DelayP is as well not completely settled. Yet, the results from Section 3.2 established valuable connections between the classical and the parametrised world which might prove themselves utilisable in this context.

From the opposite side, comparing IncFPT to OutputFPT, the following sensible first step might focus their relation more sharply. Are there natural problems in OutputFPT which do not allow for IncFPT enumeration? For this question, there might exist some candidates in the area of dependence logic which has been investigated in Chapter 6. In particular, some problems of this logic are of high complexity, and, accordingly, are good contenders for this complexity class. For

instance, validity in propositional dependence logic (PD) might be such a problem worth to study. Virtema proved the problem to be complete for nondeterministic exponential time [Vir17]. It is well-known that validity (or the tautology problem) in plain propositional logic is coNP-complete [Coo71; Lev73]. However, under team semantics, the complexity of this problem severely increases as one requires all teams (of which generally 2^{2^n} many exist for n variables) to satisfy the given formula.

Concluding, further investigating the class incremental FPT time, IncFPT, might even yield transferable results to incremental polynomial time, IncP, as we have seen how the classical and parametrised world relate in Chapter 3. Also, it would be engaging to study connections from parametrised enumeration to proof theory via the work of Goldberg and Papadimitriou [GP17]. It is important to search for intermediate natural problems between $\mathsf{F}(\mathsf{FPT})$ and $\mathsf{TF}(\mathsf{para\text{-}NP})$ which are relevant in some area beyond the trivial parametrisation $\kappa_{\text{one}}(x) = 1$.

Last to mention, in 2017, Grohe and Schweikhardt [GS17] consider FO-query evaluation with cardinality conditions. In their investigations, they mention the complexity class FPL, that is defined as fixed-parameter linear encompassing algorithms with a runtime of $f(\kappa(x)) \cdot |x|$. For instance, model-checking and counting for FO is in FPL for classes of bounded tree-width [ALS91; Cou90], of bounded degree [Fri04; See96] or bounded expansion [DKT10; KS13], and also for planar graphs [FG01]. We want to close with the question whether some of these problems might allow for DelayFPL algorithms?

Bibliography

[Abr+13] S. Abramsky, J. Kontinen, J. A. Väänänen, and H. Vollmer. “Dependence Logic: Theory and Applications (Dagstuhl Seminar 13071)”. In: *Dagstuhl Reports* 3.2 (2013), pp. 45–54. DOI: 10.4230/DagRep.3.2.45 (cit. on pp. 10, 73).

[Abr+16] S. Abramsky, J. Kontinen, J. Väänänen, and H. Vollmer, eds. *Dependence Logic, Theory and Applications*. Springer, 2016. DOI: 10.1007/978-3-319-31803-5 (cit. on p. 73).

[AF96] D. Avis and K. Fukuda. “Reverse Search for Enumeration”. In: *Discrete Applied Mathematics* 65.1-3 (1996), pp. 21–46. DOI: 10.1016/0166-218X(95)00026-N (cit. on pp. 6, 62).

[AFL04] A. Andersson, R. Fagerberg, and K. S. Larsen. “Balanced Binary Search Trees”. In: *Handbook of Data Structures and Applications*. Ed. by D. P. Mehta and S. Sahni. Chapman and Hall/CRC, 2004. DOI: 10.1201/9781420035179.ch10 (cit. on p. 22).

[AFN04] J. Alber, H. Fernau, and R. Niedermeier. “Parameterized complexity: exponential speed-up for planar graph problems”. In: *J. Algorithms* 52.1 (2004), pp. 26–56. DOI: 10.1016/j.jalgor.2004.03.005 (cit. on p. 4).

[ALS91] S. Arnborg, J. Lagergren, and D. Seese. “Easy Problems for Tree-Decomposable Graphs”. In: *J. Algorithms* 12.2 (1991), pp. 308–340. DOI: 10.1016/0196-6774(91)90006-K (cit. on p. 100).

[Ang+16] E. Angel, E. Bampis, B. Escoffier, and M. Lampis. “Parameterized Power Vertex Cover”. In: *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*. Ed. by P. Heggernes. Vol. 9941. Lecture Notes in Computer Science. 2016, pp. 97–108. DOI: 10.1007/978-3-662-53536-3_9 (cit. on p. 39).

[AYZ95] N. Alon, R. Yuster, and U. Zwick. “Color-coding”. In: *J. ACM* 42.4 (July 1995), pp. 844–856. DOI: 10.1145/210332.210337 (cit. on p. 4).

[Bea65] E. M. L. Beale. “Survey of Integer Programming”. In: *Journal of the Operational Research Society* 16.2 (June 1965), pp. 219–228. DOI: 10.1057/jors.1965.31 (cit. on p. 2).

[BG82] A. Blass and Y. Gurevich. “On the Unique Satisfiability Problem”. In: *Information and Control* 55.1-3 (1982), pp. 80–88. DOI: 10.1016/S0019-9958(82)90439-9 (cit. on p. 34).

[BG93] J. F. Buss and J. Goldsmith. “Nondeterminism Within P”. In: *SIAM J. Comput.* 22.3 (1993), pp. 560–572. DOI: 10.1137/0222038 (cit. on pp. 6, 39).

[BG94] M. Bellare and S. Goldwasser. “The Complexity of Decision Versus Search”. In: *SIAM J. Comput.* 23.1 (1994), pp. 97–119. DOI: 10.1137/S0097539792228289 (cit. on p. 29).

[BHJ16] T. Balyo, M. J. H. Heule, and M. J. Järvisalo, eds. *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*. University of Helsinki, Department of Computer Science. Department of Computer Science Series of Publications B, 2016 (cit. on p. 7).

[BHJ17] T. Balyo, M. J. H. Heule, and M. Järvisalo. “SAT Competition 2016: Recent Developments”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. Ed. by S. P. Singh and S. Markovitch. AAAI Press, 2017, pp. 5061–5063 (cit. on p. 7).

[BHL14] H. L. Bodlaender, P. Heggernes, and D. Lokshtanov. “Graph Modification Problems (Dagstuhl Seminar 14071)”. In: *Dagstuhl Reports* 4.2 (2014), pp. 38–59. DOI: 10.4230/DagRep.4.2.38 (cit. on p. 13).

[BK08] H. L. Bodlaender and A. M. C. A. Koster. “Combinatorial Optimization on Graphs of Bounded Treewidth”. In: *Comput. J.* 51.3 (2008), pp. 255–269. DOI: 10.1093/comjnl/bxm037 (cit. on p. 4).

[BLS88] A. Brandtstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. Monographs on Discrete Applied Mathematics. Philadelphia: SIAM, 1988. DOI: 10.1137/1.9780898719796 (cit. on p. 60).

[Bra15] J. Bradfield. “On the structure of events in Boolean games”. In: *Logics for Dependence and Independence*. Dagstuhl Reports, 2015 (cit. on p. 10).

[Bus+92] S. R. Buss, S. A. Cook, A. Gupta, and V. Ramachandran. “An Optimal Parallel Algorithm for Formula Evaluation”. In: *SIAM J. Comput.* 21.4 (1992), pp. 755–780. DOI: 10.1137/0221046 (cit. on p. 9).

[Cai96] L. Cai. “Fixed-parameter tractability of graph modification problems for hereditary properties”. In: *Information Processing Letters* 58.4 (1996), pp. 171–176. DOI: https://doi.org/10.1016/0020-0190(96)00050-6 (cit. on pp. 57–59, 65).

[CB16] N. Creignou and D. Le Berre, eds. *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. Vol. 9710. Lecture Notes in Computer Science. Springer, 2016. DOI: 10.1007/978-3-319-40970-2.

[CF12] A. Del Centina and A. Fiocca. “The correspondence between Sophie Germain and Carl Friedrich Gauss”. In: *Archive for History of Exact Sciences* 66.6 (2012), pp. 585–700. DOI: 10.1007/s00407-012-0105-x (cit. on p. 1).

[CH97] N. Creignou and J.-J. Hébrard. “On generating all solutions of generalized satisfiability problems”. In: *Theoretical Informatics and Applications* 31.6 (1997), pp. 499–511 (cit. on pp. 7, 10, 42, 45, 53, 56, 61).

[Chr76] N. Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Report 388. Graduate School of Industrial Administration, Carnegie Mellon University (CMU), 1976 (cit. on p. 3).

[CKS01] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Monographs on Discrete Applied Mathematics. SIAM Discrete Mathematics and Applications, 2001. DOI: 10.1137/1.9780898718546 (cit. on p. 51).

[Cla] Clay Mathematics Institute. *Millenium Prize Problems*. http://www.claymath.org/millennium-problems, last checked 20.03.2018. (cit. on p. 2).

[Coo71] S. A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047 (cit. on pp. 7, 100).

[COS11] N. Creignou, F. Olive, and J. Schmidt. "Enumerating All Solutions of a Boolean CSP by Non-decreasing Weight". In: *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*. Ed. by K. A. Sakallah and L. Simon. Vol. 6695. Lecture Notes in Computer Science. Springer, 2011, pp. 120–133. DOI: 10.1007/978-3-642-21581-0_11 (cit. on pp. 6, 17).

[Cou90] B. Courcelle. "The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs". In: *Inf. Comput.* 85.1 (1990), pp. 12–75. DOI: 10.1016/0890-5401(90)90043-H (cit. on p. 100).

[Cre+13] N. Creignou, A. Meier, J.-S. Müller, J. Schmidt, and H. Vollmer. "Paradigms for Parameterized Enumeration". In: *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*. Ed. by K. Chatterjee and J. Sgall. Vol. 8087. Lecture Notes in Computer Science. Springer, 2013, pp. 290–301. DOI: 10.1007/978-3-642-40313-2_27 (cit. on pp. 9, 14, 20).

[Cre+15] N. Creignou, R. Ktari, A. Meier, J.-S. Müller, F. Olive, and H. Vollmer. "Parameterized Enumeration for Modification Problems". In: *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*. Ed. by A.-H. Dediu, E. Formenti, C. Martın-Vide, and B. Truthe. Vol. 8977. Lecture Notes in Computer Science. Springer, 2015, pp. 524–536. DOI: 10.1007/978-3-319-15579-1_41 (cit. on pp. 9, 14, 27).

[Cre+17a] N. Creignou, M. Kröll, R. Pichler, S. Skritek, and H. Vollmer. "On the Complexity of Hard Enumeration Problems". In: *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden, March 6-9, 2017, Proceedings*. Ed. by F. Drewes, C. Martın-Vide, and B. Truthe. Vol. 10168. Lecture Notes in Computer Science. 2017, pp. 183–195. DOI: 10.1007/978-3-319-53733-7_13 (cit. on pp. 15, 24, 99).

[Cre+17b] N. Creignou, A. Meier, J.-S. Müller, J. Schmidt, and H. Vollmer. "Paradigms for Parameterized Enumeration". In: *Theory Comput. Syst.* 60.4 (2017), pp. 737–758. DOI: 10.1007/s00224-016-9702-4 (cit. on pp. 9, 14).

[Cre+19] Nadia Creignou, Raıda Ktari, Arne Meier, Julian-Steffen Müller, Frédéric Olive, and Heribert Vollmer. "Parameterised Enumeration for Modification Problems". In: *Algorithms* 12.9 (2019), p. 189. DOI: 10.3390/a12090189 (cit. on pp. 9, 14).

[CS17] F. Capelli and Y. Strozecki. "On The Complexity of Enumeration". In: *CoRR* 1703.01928v2 (2017) (cit. on pp. 19, 24–27, 30, 33).

[CV08] N. Creignou and H. Vollmer. "Boolean Constraint Satisfaction Problems: When Does Post's Lattice Help?" In: *Complexity of Constraints - An Overview of Current Research Themes [Result of a Dagstuhl Seminar]*. Ed. by N. Creignou, P. G. Kolaitis, and H. Vollmer. Vol. 5250. Lecture Notes in Computer Science. Springer, 2008, pp. 3–37. DOI: 10.1007/978-3-540-92800-3_2 (cit. on p. 51).

[CV15] N. Creignou and H. Vollmer. "Parameterized Complexity of Weighted Satisfiability Problems: Decision, Enumeration, Counting". In: *Fundam. Inform.* 136.4 (2015), pp. 297–316. DOI: 10.3233/FI-2015-1159 (cit. on p. 71).

[CW16a] G. Charwat and S. Woltran. *BDD-based Dynamic Programming on Tree Decompositions*. Report DBAI-TR-2016-95. DBAI, Fakultät für Informatik an der Technischen Universität Wien, 2016 (cit. on p. 4).

[CW16b] G. Charwat and S. Woltran. "Dynamic Programming-based QBF Solving". In: *Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF 2016) co-located with 19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016), Bordeaux, France, July 4, 2016*. Ed. by F. Lonsing and M. Seidl. Vol. 1719. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 27–40 (cit. on p. 4).

[Cyg+15] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. DOI: 10.1007/978-3-319-21275-3 (cit. on pp. 4, 37).

[Dam06] P. Damaschke. "Parameterized enumeration, transversals, and imperfect phylogeny reconstruction". In: *Theor. Comput. Sci.* 351.3 (2006), pp. 337–350. DOI: 10.1016/j.tcs.2005.10.004 (cit. on pp. 9, 10, 21, 38).

[DF13] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. London, UK, 2013. DOI: 10.1007/978-1-4471-5559-1 (cit. on pp. 3, 6, 15).

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. New York, NY, USA, 1999. DOI: 10.1007/978-1-4612-0515-9 (cit. on pp. 3, 4, 112).

[DHK05] A. Durand, M. Hermann, and P. G. Kolaitis. "Subtractive reductions and complete problems for counting complexity classes". In: *Theor. Comput. Sci.* 340.3 (2005), pp. 496–513. DOI: 10.1016/j.tcs.2005.03.012 (cit. on p. 24).

[DKT10] Z. Dvorak, D. Kral, and R. Thomas. "Deciding First-Order Properties for Sparse Graphs". In: *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. FOCS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 133–142. DOI: 10.1109/FOCS.2010.20 (cit. on p. 100).

[DKV16] A. Durand, J. Kontinen, and H. Vollmer. "Expressivity and complexity of dependence logic". In: *Dependence Logic: Theory and Applications*. Birkhäuser, 2016, pp. 5–32 (cit. on p. 75).

[DOS12] W. Dvořák, S. Ordyniak, and S. Szeider. "Augmenting tractable fragments of abstract argumentation". In: 186 (2012), pp. 157–173. DOI: 10.1016/j.artint.2012.03.002 (cit. on p. 42).

[DS11] A. Durand and Y. Strozecki. “Enumeration Complexity of Logical Query Problems with Second-order Variables”. In: *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*. Ed. by M. Bezem. Vol. 12. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011, pp. 189–202. DOI: 10.4230/LIPIcs.CSL.2011.189 (cit. on p. 7).

[EL12] J. Ebbing and P. Lohmann. “Complexity of Model Checking for Modal Dependence Logic”. In: *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21-27, 2012. Proceedings*. Ed. by M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán. Vol. 7147. Lecture Notes in Computer Science. Springer, 2012, pp. 226–237. DOI: 10.1007/978-3-642-27660-6_19 (cit. on p. 73).

[Elf+16] J. Elffers, J. Johannsen, M. Lauria, T. Magnard, J. Nordström, and M. Vinyals. “Trade-offs Between Time and Memory in a Tighter Model of CDCL SAT Solvers”. In: *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. Ed. by N. Creignou and D. Le Berre. Vol. 9710. Lecture Notes in Computer Science. Springer, 2016, pp. 160–176. DOI: 10.1007/978-3-319-40970-2_11 (cit. on p. 7).

[Enc17] C. Enchelmaier. *Route planning: using algorithms to make the most of data*. https://binando.com/blog/route-planning-using-algorithms-to-make-the-most-of-data, last checked 21.03.2018. 2017 (cit. on p. 3).

[Fer02] H. Fernau. “On Parameterized Enumeration”. In: *Computing and Combinatorics, 8th Annual International Conference, COCOON 2002, Singapore, August 15-17, 2002, Proceedings*. Ed. by O. H. Ibarra and L. Zhang. Vol. 2387. Lecture Notes in Computer Science. Springer, 2002, pp. 564–573. DOI: 10.1007/3-540-45655-4_60 (cit. on pp. 9, 21).

[FG01] M. Frick and M. Grohe. “Deciding First-order Properties of Locally Tree-decomposable Structures”. In: *J. ACM* 48.6 (Nov. 2001), pp. 1184–1206. DOI: 10.1145/504794.504798 (cit. on p. 100).

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. DOI: 10.1007/3-540-29953-X (cit. on pp. 6, 15, 17, 21, 23, 24, 38).

[Fic+17] J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. “Answer Set Solving with Bounded Treewidth Revisited”. In: *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*. Ed. by M. Balduccini and T. Janhunen. Vol. 10377. Lecture Notes in Computer Science. Springer, 2017, pp. 132–145. DOI: 10.1007/978-3-319-61660-5_13 (cit. on p. 4).

[FL97] M. Frances and A. Litman. “On covering problems of codes”. In: *Theory Comput. Syst.* 30.2 (1997), pp. 113–119. DOI: 10.1007/s002240000044 (cit. on p. 68).

[FMS16] J. K. Fichte, A. Meier, and I. Schindler. “Strong Backdoors for Default Logic”. In: *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. Ed. by N. Creignou and D. Le Berre. Vol. 9710. Lecture Notes in Computer Science. Springer, 2016, pp. 45–59. DOI: 10.1007/978-3-319-40970-2_4 (cit. on p. 42).

[Fri04] M. Frick. “Generalized Model-Checking over Locally Tree-Decomposable Classes”. In: *Theory Comput. Syst.* 37.1 (2004), pp. 157–191. DOI: 10.1007/s00224-003-1111-9 (cit. on p. 100).

[FS15a] J. K. Fichte and S. Szeider. “Backdoors to Normality for Disjunctive Logic Programs”. In: *ACM Trans. Comput. Log.* 17.1 (2015), 7:1–7:23. DOI: 10.1145/2818646 (cit. on p. 42).

[FS15b] J. K. Fichte and S. Szeider. “Backdoors to tractable answer set programming”. In: *Artif. Intell.* 220 (2015), pp. 64–103. DOI: 10.1016/j.artint.2014.12.001 (cit. on p. 42).

[FSV13] F. V. Fomin, S. Saurabh, and Y. Villanger. “A Polynomial Kernel for Proper Interval Vertex Deletion”. In: *SIAM Journal Discrete Mathematics* 27.4 (2013), pp. 1964–1976. DOI: 10.1137/12089051X (cit. on pp. 10, 38).

[Gal12] P. Galliani. “Inclusion and exclusion dependencies in team semantics - On some logics of imperfect information”. In: *Ann. Pure Appl. Logic* 163.1 (2012), pp. 68–84. DOI: 10.1016/j.apal.2011.08.005 (cit. on p. 11).

[Gas+17] S. Gaspers, N. Misra, S. Ordyniak, S. Szeider, and S. Zivny. “Backdoors into heterogeneous classes of SAT and CSP”. In: *J. Comput. Syst. Sci.* 85 (2017), pp. 38–56. DOI: 10.1016/j.jcss.2016.10.007 (cit. on p. 42).

[GJ90] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990 (cit. on p. 57).

[GNR03] J. Gramm, R. Niedermeier, and P. Rossmanith. “Fixed-Parameter Algorithms for CLOSEST STRING and Related Problems”. In: *Algorithmica* 37.1 (2003), pp. 25–42. DOI: 10.1007/s00453-003-1028-3 (cit. on p. 68).

[GP17] P. W. Goldberg and C. H. Papadimitriou. “Towards a Unified Complexity Theory of Total Functions”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 24 (2017), p. 56 (cit. on pp. 30, 100).

[GS12] S. Gaspers and S. Szeider. “Backdoors to Satisfaction”. In: *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*. Ed. by H. L. Bodlaender, R. Downey, F. V. Fomin, and D. Marx. Vol. 7370. Lecture Notes in Computer Science. Springer, 2012, pp. 287–317. DOI: 10.1007/978-3-642-30891-8_15 (cit. on p. 69).

[GS17] M. Grohe and N. Schweikardt. “First-Order Query Evaluation with Cardinality Conditions”. In: *CoRR* abs/1707.05945 (2017). arXiv: 1707.05945 (cit. on p. 100).

[GSS02] G. Gottlob, F. Scarcello, and M. Sideri. “Fixed-parameter complexity in AI and nonmonotonic reasoning”. In: *Artif. Intell.* 138.1-2 (2002), pp. 55–86. DOI: 10.1016/S0004-3702(02)00182-0 (cit. on p. 4).

[GV13] E. Grädel and J. A. Väänänen. “Dependence and Independence”. In: *Studia Logica* 101.2 (2013), pp. 399–410. DOI: 10.1007/s11225-013-9479-2 (cit. on p. 11).

[Ham53] W. R. Hamilton. “Account of the Icosian Game”. In: *Proc. Roy. Irish. Acad.* 6 (1853), pp. 415–416 (cit. on p. 1).

[Hem16] Lane A. Hemaspaandra. “SIGACT News Complexity Theory Column 90: Introduction to Complexity Theory Column 90”. In: *SIGACT News* 47.1 (Mar. 2016), pp. 41–41. DOI: 10.1145/2902945.2902956 (cit. on p. 23).

[Hen59] L. Henkin. “Some remarks on infinitely long formulas”. In: *Infinitistic methods, Proceedings of the Symposium on Foundations of Mathematics*. Państwowe Wydawnictwo Naukowe, Warsaw, and Pergamon Press, Oxford-London-New York-Paris, 1959, pp. 167–183 (cit. on p. 10).

[HNW08] F. Hüffner, R. Niedermeier, and S. Wernicke. “Techniques for Practical Fixed-Parameter Algorithms”. In: *The Computer Journal* 51.1 (2008), pp. 7–25. DOI: 10.1093/comjnl/bxm040 (cit. on p. 4).

[Hod97a] W Hodges. “Compositional Semantics for a Language of Imperfect Information”. In: *Logic Journal of the IGPL* 5.4 (1997), pp. 539–563. DOI: 10.1093/jigpal/5.4.539 (cit. on pp. 10, 73).

[Hod97b] W. Hodges. “Some Strange Quantifiers”. In: *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht*. London, UK, UK: Springer-Verlag, 1997, pp. 51–65 (cit. on p. 10).

[HS65] J. Hartmanis and R. E. Stearns. “On the computational complexity of algorithms”. In: *Trans. Amer. Math. Soc.* 117 (1965), pp. 285–306. DOI: 10.1090/S0002-9947-1965-0170805-7 (cit. on p. 27).

[HS89] J. Hintikka and G. Sandu. “Informational Independence as a Semantical Phenomenon”. In: *Logic, Methodology and Philosophy of Science VIII*. Ed. by Jens Erik Fenstad, Ivan T. Frolov, and Risto Hilpinen. Vol. 126. Studies in Logic and the Foundations of Mathematics Supplement C. Elsevier, 1989, pp. 571–589. DOI: https://doi.org/10.1016/S0049-237X(08)70066-1 (cit. on p. 10).

[JPY88a] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. “How Easy is Local Search?” In: *J. Comput. Syst. Sci.* 37.1 (1988), pp. 79–100. DOI: 10.1016/0022-0000(88)90046-3 (cit. on p. 29).

[JPY88b] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. “On Generating All Maximal Independent Sets”. In: *Inf. Process. Lett.* 27.3 (1988), pp. 119–123. DOI: 10.1016/0020-0190(88)90065-8 (cit. on pp. 6, 17, 19, 21).

[Kar72] R. M. Karp. “Reducibility Among Combinatorial Problems”. In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.* Ed. by R. E. Miller and J. W. Thatcher. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103 (cit. on pp. 2, 5).

[Kha+05] L. G. Khachiyan, E. Boros, K. M. Elbassioni, V. Gurvich, and K. Makino. “On the Complexity of Some Enumeration Problems for Matroids”. In: *SIAM J. Discrete Math.* 19.4 (2005), pp. 966–984. DOI: 10.1137/S0895480103428338 (cit. on p. 9).

[KMW16] S. Kratsch, D. Marx, and M. Wahlström. “Parameterized Complexity and Kernelizability of Max Ones and Exact Ones Problems”. In: vol. 8. 1. 2016, 1:1–1:28. DOI: 10.1145/2858787 (cit. on pp. 42–44).

[Knu98] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. 2nd. Addison-Wesley, 1998 (cit. on p. 85).

[Kon13] J. Kontinen. “Coherence and Computational Complexity of Quantifier-free Dependence Logic Formulas”. In: *Studia Logica* 101.2 (Apr. 2013), pp. 267–291. DOI: 10.1007/s11225-013-9481-8 (cit. on p. 83).

[KOP15] M. Kronegger, S. Ordyniak, and A. Pfandler. “Variable-Deletion Backdoors to Planning”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by B. Bonet and S. Koenig. AAAI Press, 2015, pp. 3305–3312 (cit. on p. 42).

[KS13] W. Kazana and L. Segoufin. “Enumeration of First-order Queries on Classes of Structures with Bounded Expansion”. In: *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. PODS ’13. New York, New York, USA: ACM, 2013, pp. 297–308. DOI: 10.1145/2463664.2463667 (cit. on p. 100).

[KST99] H. Kaplan, R. Shamir, and R. E. Tarjan. “Tractability of Parameterized Completion Problems on Chordal, Strongly Chordal, and Proper Interval Graphs”. In: *SIAM J. Comput.* 28.5 (1999), pp. 1906–1922. DOI: 10.1137/S0097539796303044 (cit. on pp. 59, 65).

[KV91] S. Khuller and V. V. Vazirani. “Planar Graph Coloring is not Self-Reducible, Assuming P $\neq$ NP”. In: *Theor. Comput. Sci.* 88.1 (1991), pp. 183–189. DOI: 10.1016/0304-3975(91)90081-C (cit. on pp. 10, 37).

[Lev73] L. A. Levin. “Universal sorting problems”. In: *Problems of Information Transmission* 9 (1973), pp. 265–266 (cit. on pp. 7, 100).

[Lip+78] R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, and A. V. Aho, eds. *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*. ACM, 1978.

[Loh12] P. Lohmann. “Computational Aspects of Dependence Logic”. PhD thesis. Leibniz Universität Hannover, Institut für Theoretische Informatik, 2012. arXiv: 1206.4564 (cit. on p. 11).

[LR18] R. J. Lipton and K. W. Regan. *The Lemma Cited From Burnside*. https://rjlipton.wordpress.com/2018/03/03/the-lemma-cited-from-burnside/, last checked 10.04.2018. 2018 (cit. on p. 76).

[LV13] P. Lohmann and H. Vollmer. “Complexity Results for Modal Dependence Logic”. In: *Studia Logica* 101.2 (2013), pp. 343–366. DOI: 10.1007/s11225-013-9483-6 (cit. on p. 73).

[Mar+02] H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. “Cutting planes in integer and mixed integer programming”. In: *Discrete Applied Mathematics* 123.1 (2002), pp. 397–446. DOI: https://doi.org/10.1016/S0166-218X(01)00348-1 (cit. on p. 2).

[Mar05] D. Marx. “Parameterized complexity of constraint satisfaction problems”. In: *Computational Complexity* 14.2 (2005), pp. 153–183. DOI: 10.1007/s00037-005-0195-9 (cit. on p. 71).

[Mei+16] A. Meier, S. Ordyniak, R. Sridharan, and I. Schindler. “Backdoors for Linear Temporal Logic”. In: *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*. Ed. by J. Guo and D. Hermelin. Vol. 63. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 23:1–23:17. DOI: 10.4230/LIPIcs.IPEC.2016.23 (cit. on p. 42).

[Mei18] A. Meier. “Enumeration in Incremental FPT-Time”. In: *CoRR* abs/1804.07799 (2018). arXiv: 1804.07799 (cit. on p. 14).

[Men87] E. Mendelson. *Introduction to Mathematical Logic*. Springer, Boston, MA, 1987. DOI: 10.1007/978-1-4615-7288-6 (cit. on p. 15).

[MP91] N. Megiddo and C. H. Papadimitriou. “On Total Functions, Existence Theorems and Computational Complexity”. In: *Theor. Comput. Sci.* 81.2 (1991), pp. 317–324. DOI: 10.1016/0304-3975(91)90200-L (cit. on pp. 14, 28–30, 33).

[MR17] A. Meier and C. Reinbold. “Enumeration Complexity of Poor Man’s Propositional Dependence Logic”. In: *CoRR* abs/1704.03292 (2017) (cit. on p. 14).

[MR18] A. Meier and C. Reinbold. “Enumeration Complexity of Poor Man’s Propositional Dependence Logic”. In: *Foundations of Information and Knowledge Systems - 10th International Symposium, FoIKS 2018, Budapest, Hungary, May 14-18, 2018. Proceedings*. Ed. by F. Ferrarotti and S. Woltran. Lecture Notes in Computer Science. Springer, 2018 (cit. on pp. 14, 27).

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006 (cit. on pp. 6, 15).

[Nor15] J. Nordström. “On the interplay between proof complexity and SAT solving”. In: *SIGLOG News* 2.3 (2015), pp. 19–44. DOI: 10.1145/2815493.2815497 (cit. on p. 7).

[NRS04] N. Nishimura, P. Ragde, and S. Szeider. “Detecting Backdoor Sets with Respect to Horn and Binary Clauses”. In: *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*. 2004 (cit. on pp. 47, 48).

[NRS07] Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. “Solving #SAT using vertex covers”. In: *Acta Informatica* 44.7-8 (2007), pp. 509–523. DOI: 10.1007/s00236-007-0056-x (cit. on p. 70).

[NT75] G. L. Nemhauser and L. E. Trotter Jr. “Vertex packings: Structural properties and algorithms”. In: *Math. Program.* 8.1 (1975), pp. 232–248. DOI: 10.1007/BF01580444 (cit. on p. 6).

[NZ09] G. Nordh and B. Zanuttini. “Frozen Boolean Partial Co-clones”. In: *ISMVL 2009, 39th International Symposium on Multiple-Valued Logic, 21-23 May 2009, Naha, Okinawaw, Japan*. IEEE Computer Society, 2009, pp. 120–125. DOI: 10.1109/ISMVL.2009.10 (cit. on p. 44).

[OC03] R. O’Callahan and J.-D. Choi. “Hybrid dynamic data race detection”. In: *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2003, June 11-13, 2003, San Diego, CA, USA*. Ed. by R. Eigenmann and M. C. Rinard. ACM, 2003, pp. 167–178. DOI: 10.1145/781498.781528 (cit. on p. 6).

[Per03] G. Perelman. “Ricci flow with surgery on three-manifolds”. In: *CoRR* (2003). arXiv: `math/0303109` (cit. on p. 2).

[Pos41] E. Post. “The two-valued iterative systems of mathematical logic”. In: *Annals of Mathematical Studies* 5 (1941), pp. 1–122 (cit. on p. 71).

[PRS13] A. Pfandler, S. Rümmele, and S. Szeider. “Backdoors to Abduction”. In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI’13)*. Ed. by Francesca Rossi. Beijing, China, Aug. 2013, pp. 1046–1052 (cit. on p. 42).

[PV06] G. Pan and M. Y. Vardi. “Fixed-Parameter Hierarchies inside PSPACE”. In: *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*. IEEE Computer Society, 2006, pp. 27–36. DOI: `10.1109/LICS.2006.25` (cit. on p. 4).

[Rot95] J. J. Rotman. *An Introduction to the Theory of Groups*. Vol. 148. Graduate Texts in Mathematics. Springer, 1995 (cit. on pp. 75, 76).

[RSV04] B. Reed, K. Smith, and A. Vetta. “Finding odd cycle transversals”. In: *Operations Research Letters* 32.4 (2004), pp. 299–301. DOI: `https://doi.org/10.1016/j.orl.2003.10.009` (cit. on p. 4).

[RT75] R. C. Read and R. E. Tarjan. “Bounds on backtrack algorithms for listing cycles, paths, and spanning trees”. In: *Networks* 5.3 (1975), pp. 237–252 (cit. on p. 7).

[SAT] SAT Association. *Website of the SAT association*. `https://satassociation.org`, last checked 09.01.2018. (cit. on p. 7).

[Sch09] J. Schmidt. “Enumeration: Algorithms and Complexity”. Online at `https://www.thi.uni-hannover.de/fileadmin/forschung/arbeiten/schmidt-da.pdf`. MA thesis. Leibniz Universität Hannover, 2009 (cit. on pp. 10, 17, 19, 20, 24, 37).

[Sch76] C.-P. Schnorr. “Optimal Algorithms for Self-Reducible Problems”. In: *Proceedings International Colloquium on Automata, Languages, and Programming*. 1976, pp. 322–337 (cit. on pp. 10, 37).

[Sch78] T. J. Schaefer. “The Complexity of Satisfiability Problems”. In: *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*. Ed. by R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, and A. V. Aho. ACM, 1978, pp. 216–226. DOI: `10.1145/800133.804350` (cit. on p. 42).

[See96] D. Seese. “Linear Time Computable Problems and First-Order Descriptions”. In: *Mathematical Structures in Computer Science* 6.6 (1996), pp. 505–526 (cit. on p. 100).

[Shp15] I. Shpitser. “Causal Inference and Logics of Dependence and Independence”. In: *Logics for Dependence and Independence*. Dagstuhl Reports, 2015 (cit. on p. 10).

[Sip12] Michael Sipser. *Introduction to the Theory of Computation*. 3rd. Cengage Learning, 2012 (cit. on p. 15).

[SS08] M. Samer and S. Szeider. “Backdoor Trees”. In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. Ed. by D. Fox and C. P. Gomes. AAAI Press, 2008, pp. 363–368 (cit. on p. 38).

[SS09] M. Samer and S. Szeider. “Fixed-Parameter Tractability”. In: *Handbook of Satisfiability*. Ed. by A. Biere, M. Heule, H. van Maaren, and T. Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 425–454. DOI: 10.3233/978-1-58603-929-5-425 (cit. on p. 4).

[SST04] R. Shamir, R. Sharan, and D. Tsur. “Cluster graph modification problems”. In: *Discrete Applied Mathematics* 144.1-2 (2004), pp. 173–182. DOI: 10.1016/j.dam.2004.01.007 (cit. on p. 60).

[ST08] C. Sloper and J. A. Telle. “An Overview of Techniques for Designing Parameterized Algorithms”. In: *Comput. J.* 51.1 (2008), pp. 122–136. DOI: 10.1093/comjnl/bxm038 (cit. on p. 4).

[SV14] A. Sebö and J. Vygen. “Shorter tours by nicer ears: 7/5-Approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs”. In: *Combinatorica* 34.5 (2014), pp. 597–629. DOI: 10.1007/s00493-014-2960-3 (cit. on p. 3).

[Sze09] Stefan Szeider. “Matched Formulas and Backdoor Sets”. In: *Journal on Satisfiability, Boolean Modeling and Computation* 6.1-3 (2009), pp. 1–12 (cit. on p. 70).

[Tul04] T. Tulenheimo. “Independence-Friendly Modal Logic: Studies in its Expressive Power and Theoretical Relevance”. PhD thesis. Philosophical Studies from the University of Helsinki 4, 2004 (cit. on p. 10).

[Vää07] J. Väänänen. *Dependence Logic - A New Approach to Independence Friendly Logic*. Vol. 70. London Mathematical Society student texts. Cambridge University Press, 2007 (cit. on pp. 11, 73).

[Vää08] J. Väänänen. “Modal Dependence Logic”. In: *New Perspectives on Games and Interaction*. Ed. by K. Apt and R. van Rooij. Amsterdam University Press, 2008, pp. 237–254 (cit. on pp. 11, 73).

[Vir17] Jonni Virtema. “Complexity of validity for propositional dependence logics”. In: *Inf. Comput.* 253 (2017), pp. 224–236. DOI: 10.1016/j.ic.2016.07.008 (cit. on p. 100).

[WGS03] R. Williams, C. P. Gomes, and B. Selman. “Backdoors To Typical Case Complexity”. In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*. Ed. by G. Gottlob and T. Walsh. Morgan Kaufmann, 2003, pp. 1173–1178 (cit. on pp. 42, 47).

[Yan78] M. Yannakakis. “Node- and Edge-Deletion NP-Complete Problems”. In: *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*. Ed. by R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, and A. V. Aho. ACM, 1978, pp. 253–264. DOI: 10.1145/800133.804355 (cit. on p. 60).

[Yan81] M. Yannakakis. “Computing the minimum fill-in is NP-complete”. In: *SIAM Journal on Algebraic Discrete Methods* 2.1 (1981), pp. 77–79. DOI: 10.1137/0602010 (cit. on p. 59).

> *Parameterized complexity is based on a deal with the devil of intractability.*
>
> — **Rod G. Downey and Michael R. Fellows**
> [DF99, p. 7]

Colophon

This thesis was typeset with LuaLaTeX. It uses a modification of the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.
Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Erklärung

Ich erkläre an Eides statt, dass ich die Arbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass bei der Anfertigung kein wissenschaftliches Fehlverhalten im Sinne der Richtlinie der Gottfried Wilhelm Leibniz Universität Hannover zur Sicherung guter wissenschaftlicher Praxis vorliegt.

I declare under penalty of perjury that this thesis is my own work entirely and has been written without any help from other people. I used only the sources mentioned and included all the citations correctly both in word or content. Furthermore, I declare that no academic misconduct in accordance with the regulations of the Gottfried Wilhelm Leibniz Universität Hannover exists.

Hannover, 20.11.2019

Arne Meier