algorithms

MDPI

Article

Incremental FPT Delay

Arne Meier

Institut für Theoretische Informatik, Leibniz Universität Hannover, 30167 Hannover, Germany; meier@thi.uni-hannover.de; Tel.: +49-(0)511-762-19768

Received: 10 March 2020; Accepted: 12 May 2020; Published: 15 May 2020

Abstract: In this paper, we study the relationship of parameterized enumeration complexity classes defined by Creignou et al. (MFCS 2013). Specifically, we introduce two hierarchies (IncFPTa and CapIncFPTa) of enumeration complexity classes for incremental fpt-time in terms of exponent slices and show how they interleave. Furthermore, we define several parameterized function classes and, in particular, introduce the parameterized counterpart of the class of nondeterministic multivalued functions with values that are polynomially verifiable and guaranteed to exist, TFNP, known from Megiddo and Papadimitriou (TCS 1991). We show that this class TF(para-NP), the restriction of the function variant of NP to total functions, collapsing to F(FPT), the function variant of FPT, is equivalent to the result that OutputFPT coincides with IncFPT. In addition, these collapses are shown to be equivalent to TFNP = FP, and also equivalent to P equals NP intersected with coNP. Finally, we show that these two collapses are equivalent to the collapse of IncP and OutputP in the classical setting. These results are the first direct connections of collapses in parameterized enumeration complexity to collapses in classical enumeration complexity, parameterized function complexity, classical function complexity, and computational complexity theory.

Keywords: parameterized complexity; function complexity; enumeration

1. Introduction

1.1. Enumeration

In 1988, Johnson, Papadimitriou, and Yannakakis [1] laid the cornerstone of enumeration algorithms. In modern times of ubiquitous computing, such algorithms are of central importance in several areas of life and research such as combinatorics, computational geometry, and operations research [2]. Beyond that, recent results unveil major importance in web search, data mining, bioinformatics, and computational linguistics [3]. Moreover, there are connections to formal languages on enumeration problems for probabilistic automata [4].

Clearly, for enumeration algorithms, one is not primarily interested in their runtime complexity, whereas the time elapsed between two consecutive outputs of such an algorithm is of utmost interest. As a result, one measures the *delay* of such algorithms. The main goal is to achieve a uniform stream of printed solutions. In this context, the complexity class DelayP, that is, polynomial delay, is regarded as an efficient way of enumeration. Enumeration algorithms for problems in that particular class print every $p(|x|)$-many-steps a new and distinct solution (where x is the input and p is a fixed polynomial). Interestingly, there is exists a class of *incremental polynomial delay*, IncP, which contains problems that allow for enumeration algorithms whose delay increases in the process of computation. Intuitively, this captures

the idea that after printing 'obvious' solutions, later in the process, it becomes difficult to find new outputs. More precisely, the delay between output i and $i+1$ is bounded by a polynomial in the input length *and in* i. Consequently, in the beginning, such an algorithm possesses a polynomial delay, whereas later, it eventually becomes exponential (for problems with exponentially many solutions, which is rather a common phenomenon). While prominent problems in the class DelayP are the enumeration of satisfying assignments for Horn or Krom formulas [5], the enumeration of structures for first-order query problems with possibly free second-order variables and at most one existential quantifier [6], or the enumeration of cycles in graphs [7], rather a limited amount of research has been invested in understanding IncP. A well-studied problem in this enumeration complexity class is the task of generating all maximal solutions of systems of equations modulo 2 [8]. Even today, it is not clear whether this problem can be solved with a polynomial delay. Another example of problems in IncP is the task to enumerate conjuncts in the context of monotone dualization of DNF formulas investigated by Eiter et al. [9], and Fredman and Khachiyan [10]. The significance of these results stems from the connection to computing minimal hypergraph transversals. A more recent result, which is not known to be in DelayP and not equivalent to the enumeration of transversals in hypergraphs of fixed edge size, was shown to be a member of IncP by Carmeli et al. [11]: the task to enumerate all minimal triangulations of a given graph. In 2017, Capelli and Strozecki [12] investigated IncP and its relationship to other classical enumeration classes in depth, greatly improving the overall understanding of incremental polynomial delay. One of their central results for this paper shows that if the function variant of NP restricted to total functions coincides with FP, then we have a collapse of incremental polynomial time with OutputP, the class of problems for which all solutions can be enumerated in time polynomial in the input length and in the solution set size (see Proposition 4 on page 4). Furthermore, Capelli and Strozecki studied exponent sliced versions of incremental time and showed that this hierarchy of classes strictly interleaves ([12], Proposition 16).

1.2. Parameterized Complexity

The framework of parameterized complexity [13–16] allows one to approach a fine-grained complexity analysis of problems beyond classical worst-case complexity. Here, one considers a problem together with a parameter and tries to achieve deterministic runtimes of the form $f(\kappa(x)) \cdot p(|x|)$, where $\kappa(x)$ is the value of the parameter of an instance x (note that the parameter does not depend on the input length), p is a polynomial, and f is an arbitrary computable function. Problems that can be solved by algorithms within these runtime bounds belong to the class FPT. The class name abbreviates '*fixed-parameter tractable*' and is justified by the observation that a relevant parameter is seen be slowly growing or even of constant value [17].

A rather large parameterized complexity class is para-NP, the nondeterministic counterpart of FPT, which is defined via nondeterministic runtimes of the same form of $f(\kappa(x)) \cdot p(|x|)$. By definition of the classes, the inclusion FPT $\subseteq$ para-NP is immediate. Nonetheless, compared to the situation on the classical complexity side, para-NP is *not* seen as a counterpart of NP in the sense of intractability. In fact, W[1] is the class which usually is used to show intractability lower bounds in the parameterized setting. This class is part of an infinite W-hierarchy in between the aforementioned two classes. It is not known whether any of the inclusions of the intermediate classes is strict or not.

1.3. Parameterized Enumeration

Recently, Creignou et al. [18–21] developed a framework of parameterized enumeration allowing for fine-grained complexity analyses of enumeration problems. In analogue to classical enumeration complexity, there are the classes DelayFPT and IncFPT, and it is unknown if DelayFPT $\subsetneq$ IncFPT is true or not. Here, for problems in IncFPT, enumeration algorithms exist which print solutions with a delay of the

form $t(\kappa(x)) \cdot p(|x|, i)$, where t is a computable function, κ the parameterization, x the input, and i is the index of the currently printed solution. Similarly, problems in DelayFPT allow for enumeration algorithms running with an fpt-delay.

In their research, Creignou et al. [20] noticed that for some problems, enumerating solutions by increasing size is possible with DelayFPT and exponential space (such as triangulations of graphs, or cluster editings). However, it is not clear how to circumvent the unsatisfactory space requirement. Recently, Meier and Reinbold [22] observed a similar phenomenon. They studied the enumeration complexity of problems in a modern family of logics of dependence and independence. In the context of dependence statements, single assignments do not make sense. As a result, one introduces *team semantics* defining semantics with respect to *sets* of assignments, which are commonly called *teams*. Meier and Reinbold showed that in the process of enumerating satisfying teams for formulas of a specific *Dependence Logic* fragment, it seemed that an FPT delay required exponential space in this particular setting. While reaching polynomial space for the same problem, the price was paid by an increasing delay, IncFPT, and it was not clear how to avoid the increase of the delay while maintaining a polynomial space threshold. This is a significant question of research, and we improve the understanding of this question by pointing out connections to classical enumeration complexity where similar phenomena have been observed [12] (see also the discussion in the beginning of Section 4).

1.4. Related Work

In 1991, Megiddo and Papadimitriou [23] introduced the function complexity class TFNP and studied problems within this class. In a recent investigation, Goldberg and Papadimitriou introduced a rich theory around this complexity class that also features several aspects of proof theory [24]. Additionally, the investigations of Capelli and Strozecki [12] on probabilistic classes might yield further connections to the enumeration setting via the parameterized analogues of probabilistic computation of Chauhan and Rao [25]. Furthermore, Fichte et al. [26] studied the parameterized complexity of default logic and present in their work a parameterized enumeration algorithm outputting stable extensions. It might be worth further analyzing problems in this setting, possibly yielding IncFPT algorithms. Quite recently, Bläsius et al. [27] considered the enumeration of minimal hitting sets in lexicographical order and devised some cases which allow for DelayP-, resp., DelayFPT-algorithms. Furthermore, Pichler et al. [28] showed enumeration complexity results for problems on MSO (monadic second order logic) formulas. Finally, investigations of Mary and Strozecki [29] are related to the IncP-versus-DelayP question from the perspective of closure operations. Counting classes in the parameterized setting have been introduced by Flum and Grohe [30] as well as by McCartin [31]. Very recently, Roth and Wellnitz [32] presented a thorough study of counting and finding homomorphisms and showed how their results improve the understanding of the class #W[1] (the counting variant of W[1]). The understanding of this class has been improved also by a line of previous research, e.g., the works of Curticapean, Dell, and Roth [33] as well as of Roth and Schmitt [34] (and more recently, together with Dörfler and Wellnitz [35]). The thesis of Curticapean [36] also presents a nice overview of this field.

1.5. Our Contribution

We improve the understanding of incremental enumeration time by connecting classical enumeration complexity to the very young field of parameterized enumeration complexity. Although we cannot answer the aforementioned time–space tradeoff question in either a positive or negative way, we hope that the presented "bridge" to parameterized enumeration will be helpful for future research. Capelli and Strozecki [12] distinguished two kinds of incremental polynomial time enumeration, which we call IncP and CapIncP. Essentially, the difference of these two classes lies in the perspective of the delay.

For IncP, one measures the delay between an output solution i and $i+1$, which has to be polynomial in i and the input length. For CapIncP, the output of i solutions has to be polynomial in i and the input length. This means that for the latter class, the output stream must not be as uniform as for the former. In Section 4, we introduce several parameterized function classes (a good overview of the classes can be found in Table 1 on page 11) that are utilized to prove our main result: IncFPT $=$ OutputFPT if and only if IncP $=$ OutputP (Corollary 4). This is the first result that directly connects the classical with the parameterized enumeration setting. Through this approach, separating the classical classes then implies separating the parameterized counterparts and vice versa. Moreover, we introduce two hierarchies of parameterized incremental time IncFPT$_a$ and CapIncFPT$_a$ (where $a \in \mathbb{N}$ is a class parameter that restricts the exponent of the polynomial for the solution index) in Section 3, show how they interleave and thereby provide some new insights into the interplay of FPT delay and incremental FPT delay. One of the previously mentioned parameterized function classes, TF(para-NP), is a counterpart of the class TFNP, the class of nondeterministic multivalued functions with values that are polynomially verifiable and guaranteed to exist, known from Megiddo and Papadimitriou [23]. For this class, the algorithmic task is to guess a witness y such that $(x, y) \in A$, where A is a total relation. It is important to note that this class summarizes significant cryptography-related problems such as factoring or the discrete logarithm modulo a (certified) prime p of a (certified) primitive root x of p (see the textbook of Talbot and Welsh [37] on cryptography-related problems). Clearly, parameterized versions of these problems are members in TF(para-NP) via trivial parameterization $\kappa_{\text{one}}(x) = 1$.

Boiling down the presented results yields a list of equivalent collapse results and thereby connects a wealth of different fields of research: parameterized enumeration, classical enumeration, parameterized function complexity, classical function complexity, and computational complexity theory. Our second central result is summarized in Corollary 3 on page 15.

1.6. Outline

In Section 2, we introduce the necessary notions of parameterized complexity theory and (parameterized) enumeration algorithms. Then, we continue in Section 3 to present two hierarchies of parameterized incremental FPT enumeration classes and study the relation to DelayFPT. Eventually, in Section 4, we introduce several parameterized function classes and outline connections to the parameterized enumeration classes. There, we connect a collapse of the two function classes TF(para-NP) and F(FPT) to a collapse of OutputFPT and CapIncFPT, extend this collapse to TFNP and FP (thus, in the classical function complexity setting), and further reach out for our main result showing OutputFPT $=$ IncFPT if and only if OutputP $=$ IncP. Finally, we conclude and present questions for future research.

2. Preliminaries

Enumeration algorithms are usually running in exponential time as the solution space is of this particular size. 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, for instance, the work of Johnson et al. [1] or, more recently, of Creignou et al. [38]. For our purposes, polynomially restricted RAMs (RAMs where each register content is polynomially bounded with respect to the input size) suffice. We make use of the standard complexity classes P and NP. An introduction into this field can be found in the textbook of Pippenger [39].

2.1. Parameterized Complexity Theory

We present a brief introduction into the field of parameterized complexity theory. For a deeper introduction, we kindly refer the reader to the textbooks of Flum and Grohe [15], Niedermeyer [16], or Downey and Fellows [13].

Let $Q \subseteq \Sigma^*$ be a decision problem over some alphabet Σ. Given an instance $\langle x, k \rangle \in Q \times \mathbb{N}$, we call k the *parameter's value (of x)*. Often, instead of using tuple notation for instances, one uses a polynomial time computable function $\kappa \colon \Sigma^* \to \mathbb{N}$ (the *parameterization*) to address the parameter's value of an input x. Then, we write (Q, κ) denoting the *parameterized problem* (PP).

Definition 1 (Fixed-parameter tractable)**.** *Let (Q, κ) be a parameterized 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 $f(\kappa(x)) \cdot |x|^{O(1)}$,*

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

Flum and Grohe [15] provide a way to "parameterize" a classical and *robust* complexity class. For our purposes, para-NP suffices, and accordingly, we do not present the general scheme.

Definition 2 (para-NP, ([15], Definition 2.10))**.** *Let (Q, κ) with $Q \subseteq \Sigma^*$ be a parameterized 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 whether $x \in Q$ in at most $f(\kappa(x)) \cdot p(|x|)$ steps, where p is a polynomial.*

Furthermore, Flum and Grohe characterize the class para-NP via all problems "*that are in* NP *after precomputation on the parameter*".

Proposition 1 ([15], Proposition 2.12)**.** *Let (Q, κ) be a parameterized problem over some alphabet Σ. We have $(Q, \kappa) \in$ para-NP if and only if there exists a computable function $\pi \colon \Sigma^* \to \Sigma^*$ 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 characterization of the complexity class NP via a verifier language, one can easily deduce the following corollary, which is later utilized to explain Definition 11.

Corollary 1. *Let (Q, κ) be a parameterized problem over some alphabet Σ and p some polynomial. We have $(Q, \kappa) \in$ para-NP if there exists a computable function $\pi \colon \Sigma^* \to \Sigma^*$ 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 $(x, \pi(\kappa(x)), y) \in Q'$.*

2.2. Enumeration

As already motivated in the beginning of this section, measuring the runtime of enumeration algorithms is usually abandoned beyond the study of OutputP. As a result, one inspects the uniformity of the flow of output solutions of these algorithms rather than their total running time. In view of this, one measures the delay between two consecutive outputs. Johnson et al. [1] laid the cornerstone of this intuition in a seminal paper and introduced the necessary tools and complexity notions. Creignou, Olive,

and Schmidt [3,40] presented recent notions in this framework, which we aim to follow. In this paper, we only consider enumeration problems which are "good" in the sense of polynomially bounded solution lengths and polynomially verifiable solutions (Capelli and Strozecki [12] call the corresponding class EnumP). A good textbook that introduces into the field of exact exponential algorithms is by Fomin and Kratsch [41].

Definition 3 (NP Enumeration Problem)**.** *An* NP enumeration problem *(EP) over an alphabet* Σ *is a tuple* $E = (Q, \mathrm{Sol})$*, where*

1. $Q \subseteq \Sigma^*$ *is the set of instances (recognizable in polynomial time),*
2. $\mathrm{Sol}\colon \Sigma^* \to \mathcal{P}(\Sigma^*)$ *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* $x \in Q$*,*
3. $\{(x, y) \mid y \in \mathrm{Sol}(x)\} \in \mathsf{P}$*, and*
4. *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 $\mathcal{S} = \bigcup_{x \in Q} \mathrm{Sol}(x)$ to refer to the set of solutions for every possible instance. If $E = (Q, \mathrm{Sol})$ is an EP over the alphabet Σ, then we call strings $x \in \Sigma^*$ *instances of* E, and $\mathrm{Sol}(x)$ the *set of solutions of* x.

An *enumeration algorithm* (EA) $\mathcal{A}$ for the EP $E = (Q, \mathrm{Sol})$ 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. The following definition fixes the ideas of measuring the 'flow of output solutions'.

Definition 4 (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. If* $t\colon \mathbb{N} \to \mathbb{N}$*, then, generally,* $\mathcal{A}$ *is said to output* $\mathrm{Sol}(x)$ *with* delay $O(t(|x|))$ *if for all* $x \in Q$ *and all* $i \in \mathbb{N}$ *the* i*-th delay is in* $O(t(|x|))$*.*

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

Definition 5. *Let* $E = (Q, \mathrm{Sol})$ *be an enumeration problem and* $\mathcal{A}$ *be an enumeration algorithm for* E*. Then,* $\mathcal{A}$ *is*

1. *a* 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* $\mathrm{Sol}(x)$ *and its* i*-th delay is in* $O(p(|x|, i))$ *(for every* $0 \leq i \leq |\mathrm{Sol}(x)|$*).*
4. *a* $\mathsf{CapIncP}_a$*-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* $O(p(|x|) \cdot i^a)$ *(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* $O(p(|x|, |\mathrm{Sol}(x)|))$*.*

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

Note that in the diploma thesis of Schmidt ([40], Section 3.1), the class P-enum is called TotalP. We avoid this name to prevent possible confusion with class names defined in the following section as well as with the work of Capelli and Strozecki [12]. Additionally, we want to point out that Capelli and Strozecki use the definition of CapIncP for IncP (and use the name "UsualIncP" for IncP instead). They prove that the notions of CapIncP and IncP are equivalent up to an exponential space requirement when using a structured delay. The idea for the following proposition is that we can wait to output previously computed solutions.

Proposition 2 ([12], Corollary 13)**.** *Without any space restrictions we have that* CapIncP = IncP*.*

After studying the definition of the class CapIncP, one could easily come to the opinion that the class is more general than IncP because it does not necessarily have a uniform delay. However, the previous proposition told us that this is not true.

2.3. Parameterized Enumeration

Having introduced the basic principles in parameterized complexity theory and enumeration complexity theory, we introduce a combined version of these notions.

Definition 6 ([19])**.** *A* parameterized enumeration problem *(PEP) over an alphabet* Σ *is a triple* $E = (Q, \kappa, \mathrm{Sol})$*, where*

- $\kappa\colon \Sigma^* \to \mathbb{N}$ *is a parameterization (that is, a polynomial-time computable function), and*
- (Q, Sol) *is an EP.*

The definitions of enumeration algorithms and delays are easily lifted to the setting of PEPs. Observe that the following defined classes are in complete analogy to the nonparameterized case from the previous section.

Definition 7 ([19])**.** *Let* $E = (Q, \kappa, \mathrm{Sol})$ *be a PEP and* $\mathcal{A}$ *an enumeration algorithm for E. Then the algorithm* $\mathcal{A}$ *is*

1. *an* FPT*-enumeration algorithm if there exists 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* $\mathrm{Sol}(x)$ *in time at most* $t(\kappa(x)) \cdot p(|x|)$*,*
2. *a* DelayFPT*-algorithm 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* $\mathrm{Sol}(x)$ *with delay of at most* $t(\kappa(x)) \cdot p(|x|)$*,*
3. *an* IncFPT*-algorithm 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* $\mathrm{Sol}(x)$ *and its i-th delay is at most* $t(\kappa(x)) \cdot p(|x|, i)$*, and*
4. *an* OutputFPT*-algorithm if there exists 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* $\mathrm{Sol}(x)$ *in time at most* $t(\kappa(x)) \cdot p(|x|, |\mathrm{Sol}(x)|)$*.*

Note that, as before, the notion of TotalFPT has been used for the class of FPT-enumerable problems [18]. We avoid this name as it causes confusion with respect to an enumeration class TotalP ([40], Section 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 nonparameterized analogue of the above class OutputFPT. Now, we group these different kinds of algorithms in complexity classes.

The class EnumFPT/DelayFPT/IncFPT/OutputFPT is the class of all PEPs that admit an FPT-enumeration/DelayFPT-/IncFPT-/OutputFPT-algorithm.

The class DelayFPT captures a good notion of tractability for parameterized enumeration complexity. Creignou et al. [19] identified a multitude of problems that admit DelayFPT enumeration. Note that, due to

Flum and Grohe ([15], Proposition 1.34) the class FPT can be characterized 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.

3. Interleaving Hierarchies of Parameterized Incremental Delay

The previous observations raise the question of how DelayFPT relates to IncFPT. In the classical enumeration world, this question has been partially answered by Capelli and Strozecki ([12], Proposition 16) for a weaker capped version of incremental polynomial time: DelayP $\subsetneq$ CapIncP is true for the restriction to solution sets that do not necessarily obey the last two items of Definition 3 (they call this class Enum · F). Only for linear total time and polynomial space under this setting is the relation not completely understood, that is, the question how DelayP with polynomial space relates to $\mathsf{CapIncP}_1$ with polynomial space ([12], Open Problem 1, p. 10).

In the course of this chapter, it is shown that the relationship between the classical and the parameterized 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 parameterized setting.

Definition 8 (Sliced Versions of Incremental FPT Delay, extending Definition 7)**.** *Let $E = (Q, \kappa, \mathrm{Sol})$ be a PEP and $\mathcal{A}$ an enumeration algorithm for E. Then, algorithm $\mathcal{A}$ is*

3′. a $\mathsf{CapIncFPT}_a$-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 $\mathrm{Sol}(x)$ and its 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 7 as the i-th delay then merely is $t(\kappa(x)) \cdot p(|x|)$, as $i^0 = 1$.

Usually, one utilizes a priority queue to store the already printed solutions as it allows for efficiently detecting duplicates. Nevertheless, it is unclear how to detect duplicates of solutions without storing them in the queue. Currently, it seems that the price of a "regular" (that is, polynomial) delay then is paid by the need for exponential space. Agreeing with Capelli and Strozecki ([12], Section 3), it seems very reasonable to see the difference of IncFPT_1 and DelayFPT anchored in the presence of an exponential sized priority queue. However, relaxing this statement shows that the equivalence of incremental FPT delay and capped incremental FPT-time is also true in the parameterized world. Similarly, as in the classical setting ([12], Proposition 12), the benefit of a structured delay can be reached with the exponential space of a priority queue. To lift the argumentation in the proof to the parameterized setting, one needs to keep track of the factors of the kind $t(\kappa(x))$ in the process. The remainder of the argumentation is the same.

Theorem 1. *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 8 (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 $\leq t(\kappa(x))i^{a+1} \cdot p(|x|)$ for all $x \in Q$, for all $0 \leq i \leq |\mathrm{Sol}(x)|$, and some computable function t (see Definition 8 (3′)). We 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). This means that we argue on the duration of the i-th delay, which summarizes the time needed to find the i-th solution and to print it. To reach this delay, one uses two counters: one for the steps of $\mathcal{A}$ (steps) and one for the solutions, initialized with value 1 (solindex). While simulating $\mathcal{A}$, the solutions are inserted into a priority queue $\mathcal{Q}$ 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 $\mathcal{Q}$ is extracted, output, and solindex is incremented by one. In view of this, $\mathcal{A}$ computed solindex many solutions and prints the next one (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 thus, $E \in \mathsf{IncFPT}_a$. □

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

Corollary 2. $\mathsf{CapIncFPT}_1 = \mathsf{DelayFPT}$ *and* $\mathsf{CapIncFPT} = \mathsf{IncFPT}$.

If one drops the restrictions 3. and 4. from Definition 3, then Capelli and Strozecki unconditionally show a strict hierarchy for the cap-classes via utilizing the well-known time hierarchy theorem [42]. Of course, this result transfers also to the parameterized world, that is, to the same generalization of $\mathsf{CapIncFPT}_a$. Nonetheless, it is unknown whether a similar hierarchy can be unconditionally shown for these classes as well as for IncFPT_a.

Open Problem 1. *Can a similar hierarchy 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 parameterized with classical enumeration complexity.

4. Connecting with Classical Enumeration Complexity

Capelli and Strozecki [12] 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 reflects with what has been observed by Creignou et al. [20]. They noticed that outputting solutions ordered by their size seems to require exponential space in case one aims for DelayFPT. Nonetheless, the mandatory order is a strong additional constraint which might weaken the evidence of that result substantially. As mentioned in the introduction, Meier and Reinbold [22] observed how a DelayFPT algorithm with exponential space of a specific problem is transformed into an IncFPT algorithm with polynomial space. These results emphasize what we strive for and why it is valuable to have such a connection between these two enumeration complexity fields. In this section, we prove that a collapse of IncP and OutputP implies OutputFPT collapsing to IncFPT and vice versa.

Capelli and Strozecki [12] 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 optimization, or binary linear programming.

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 PPs, EPs, and PEPs. Nevertheless, it is easy to see how to utilize a *witness function* f for a given language L such that $x \in L$ implies $f(x) = y$ for some y such that $A(x, y)$ is true (this means that $(x, y) \in A$ for the kind of relations mentioned before), and $f(x) =$"no" otherwise, in order to match the term "function complexity class" more adequately.

Definition 9. *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.*

Recall that, for instances of a (P)EP E over Σ, the length of its solutions is polynomially bounded. Accordingly, the underlying relation $A \subseteq \Sigma^* \times \Sigma^*$ is polynomially balanced.

The following two definitions present four function complexity classes.

Definition 10 (FP and FNP)**.** *Let $A \subseteq \Sigma^* \times \Sigma^*$ be a binary and polynomially balanced relation.*

- *$A \in$ 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.*
- *$A \in$ FNP if there is a deterministic polynomial time algorithm that can determine whether $A(x, y)$ is true, given both x and y.*

Definition 11 (F(FPT) and F(para-NP))**.** *Let $A \subseteq \Sigma^* \times \Sigma^*$ be a parameterized and polynomially balanced relation with parameterization κ.*

- *$A \in$ F(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.*
- *$A \in$ F(para-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 F(para-NP) follows the verifier characterization of precomputation on the parameter as observed in Corollary 1. Similarly to the classical decision class, NP, the runtime has to be independent of the witness length $|y|$.

The next definition lifts the class F(NP ∩ coNP) [23] to the parameterized setting. Notice that the class definition incorporates two special markers (called *a* and *b*)—similar as in the classical setting—to differentiate between the two types of witnesses.

Definition 12 ($\mathsf{F}(\mathsf{para\text{-}NP} \cap \mathsf{para\text{-}coNP})$)**.** *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 parameterized and polynomially balanced relations $A \subseteq \Sigma^* \times \Sigma^*$ and $B \subseteq \Sigma^* \times \Sigma^*$ with parameterizations 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 can find either 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 do not exist, where p, q are polynomials and f, g are computable functions.*

The typical problems in $\mathsf{F}(\mathsf{para\text{-}NP} \cap \mathsf{para\text{-}coNP})$ then ask for finding an appropriate witness (of 'type' either a or b) with respect to a given x. (Table 1)

Table 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 parameterization of A, κ' is the parameterization of B, and x is the input.

Class	Machine	Runtime	Constraints
FP	det.	$p(\|x\|)$	find y such that $A(x, y)$
FNP	nond.	$p(\|x\|)$	guess y such that $A(x, y)$
TFNP	nond.	$p(\|x\|)$	guess y such that $A(x, y)$, A is total
$\mathsf{F}(\mathsf{FPT})$	det.	$f(\kappa(x)) \cdot p(\|x\|)$	find y such that $A(x, y)$
$\mathsf{F}(\mathsf{para\text{-}NP})$	nond.	$f(\kappa(x)) \cdot p(\|x\|)$	guess y such that $A(x, y)$
$\mathsf{TF}(\mathsf{para\text{-}NP})$	nond.	$f(\kappa(x)) \cdot p(\|x\|)$	guess y such that $A(x, y)$, A is total
$\mathsf{F}(\mathsf{para\text{-}NP} \cap \mathsf{para\text{-}coNP})$	nond.	$f(\kappa(x)) \cdot p(\|x\|) + g(\kappa'(x)) \cdot q(\|x\|)$	either find y with $A(x, ay)$ or z with $B(x, bz)$

In 1994, Bellare and Goldwasser [43] 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), such functional variants are not self-reducible (as the decision variant is). Here, a problem is self-reducible means that one constructs a witness in a bit-wise fashion, given an oracle for that problem. 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 13 (Total function complexity classes)**.** TFNP *(resp.,* $\mathsf{TF}(\mathsf{para\text{-}NP})$*) is the restriction of* FNP *(resp.,* $\mathsf{F}(\mathsf{para\text{-}NP})$*) 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 $\mathsf{TF}(\mathsf{P})$ or $\mathsf{TF}(\mathsf{FPT})$ is not meaningful as it is known that $\mathsf{FP} \subseteq \mathsf{TFNP}$ (see, e.g., the work of Johnson et al. ([44], Lemma 3) showing that FP is contained in PLS, polynomial local search, which is contained in TFNP by Megiddo and Papadimitriou ([23], p. 319). Similar arguments apply to $\mathsf{F}(\mathsf{FPT}) \subseteq \mathsf{TF}(\mathsf{para\text{-}NP})$).

Now, we can define a generic (parameterized) search and a generic (parameterized) enumeration problem. Note that the parameter is only relevant for the parameterized 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 $Sol(x) \setminus S$, 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 ([12], Propositions 7 and 9) which are crucial in the course of this section are restated in the following.

Proposition 3 ([12], Proposition 7)**.** *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. Then the following is true:* ANOTHERSOL$_A \in$ FP *if and only if* ENUM-$A \in$ CapIncP.

Proposition 4 ([12], Proposition 9)**.** TFNP $=$ FP *if and only if* OutputP $=$ CapIncP.

In 1991, Megiddo and Papadimitriou studied the complexity class TFNP [23]. In a recent investigation, Goldberg and Papadimitriou introduced a rich theory around this complexity class that also features several aspects of proof theory [24]. Megiddo and Papadimitriou prove that the classes F(NP $\cap$ coNP) and TFNP coincide. Their arguments are easily lifted to the parameterized setting.

Theorem 2. F(para-NP $\cap$ para-coNP) $=$ TF(para-NP).

Proof. We reformulate the classical proofs of Megiddo and Papadimitriou [23].

"$\subseteq$": By definition of the class F(para-NP $\cap$ 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 as either of the two witnesses always exists (see Definition 12). Thus, one just needs to guess which of A or B to choose.

"$\supseteq$": the mapping from the total relation A to $(A, \emptyset)$ meets the requirements of F(para-NP $\cap$ para-coNP) as only one type of witnesses is needed, and for every x, there is a y such that $(x, y) \in A$ as A is total by assumption. □

For the following Lemma 1 (which is the parameterized pendant of Proposition 3) and theorem, we follow the argumentation in the proofs of the classical results (Propositions 3 and 4). They underline the interplay between search and enumeration.

Lemma 1. *Let $A \subseteq \Sigma^*$ be a parameterized problem with parameterization κ. Then,* para-ANOTHERSOL$_A \in$ F(FPT) *if and only if* para-ENUM-$A \in$ CapIncFPT.

Proof. "$\Rightarrow$": Let para-ANOTHERSOL$_A \in$ F(FPT) via some algorithm $\mathcal{A}$. Algorithm 1 shows that para-ENUM-$A \in$ CapIncFPT.

The runtime of each step is $f(\kappa(x)) \cdot p(|x|, |S|)$ for some polynomial p and some computable function f. Consequently, this shows that para-ENUM-$A \in$ CapIncFPT.

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

Now, we explain how to compute para-ANOTHERSOL$_A$ 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 $\mathrm{Sol}(x)$ is completely output. Just search a $y \in \mathrm{Sol}(x) \setminus S$ or output "$S \supseteq \mathrm{Sol}(x)$". Otherwise, if $\mathcal{A}$ did not halt, then it did output at least $|S|+1$ different elements. Finally, we just compute $\mathrm{Sol}(x) \setminus S$ and print a new element. □

Algorithm 1: Algorithm showing para-ENUM-$A \in$ CapIncFPT.

```
S ← ∅;
repeat
    y ← A(x, S);
    S ← S ∪ {y};
    print y;
until S = A(x);
```

The next theorem translates the result of Proposition 4 from classical enumeration complexity to the parameterized setting. It shows the first connection of parameterized enumeration to parameterized function complexity.

Theorem 3. TF(para-NP) = F(FPT) *if and only if* OutputFPT = CapIncFPT.

Proof. "⇐": Let $A \in$ TF(para-NP) be a parameterized language over $\Sigma^* \times \Sigma^*$ with parameterization κ. Furthermore, let M denote the nondeterministic algorithm that guesses the witnesses y and then verifies with the deterministic algorithm as in Definition 11. Additionally, denote the running time of M by $g(\kappa(x)) \cdot p(|x|)$ for a polynomial p, a computable function g, and an input x. Now, define the relation $C \subseteq \Sigma^* \times \{y\#w \mid y \in \Sigma^*, w \in \{0,1\}^*\}$, where $\# \notin \Sigma$, 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 the definition of the class TF(para-NP). Moreover, via padding, for each x, there exist at least $2^{p(|x|)}$ solutions z such that $C(x, z)$ is true; in particular, z is of the form $y\#w$ such that $A(x, y)$ is true. Notice that the total length of $y\#w$ is bounded by $g(\kappa(x)) \cdot p(|x|) + p(|x|) + 1$ as y already has the length constraint of $g(\kappa(x)) \cdot p(|x|)$ by the TF(para-NP) requirement. By construction, the trivial brute-force enumeration algorithm checking all $y\#w$ is in time $g(\kappa(x)) \cdot p(|x|)$ for every element of $\mathrm{Sol}(x)$. Accordingly, this gives para-ENUM-$C \in$ OutputFPT as the runtime for OutputFPT algorithms encompasses $|\mathrm{Sol}(x)|$ as a factor.

Then, para-ENUM-$C \in$ CapIncFPT and the first $y\#w$ is output in time $g(\kappa(x)) \cdot p(|x|)$. Since A was arbitrary, we conclude with TF(para-NP) = F(FPT) (as F(FPT) ⊆ TF(para-NP) by definition).

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

1. D is total by construction,
2. as para-ENUM-A is a PEP, there exists a polynomial q such that for every solution $y \in \mathrm{Sol}(x)$ we have that $|y| \leq q(|x|)$, 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 Definition 7 (4.).

Case $y = \#$: $D((x,S),y)$ is true if and only if $S \supseteq \mathrm{Sol}(x)$. As para-ENUM-$A \in$ OutputFPT, there is a deterministic 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 within this steps-bound 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 within the steps-bound, 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) = F(FPT) is true by precondition, given a tuple (x,S), we either can compute a y with $y \in \mathrm{Sol}(x) \setminus S$ or decide there is none (and then set $y = \#$) in time $f(\kappa(x)) \cdot p(|x|,|S|,|y|)$. Accordingly, para-ANOTHERSOL$_A$ is in F(FPT) and, by applying Lemma 1, we get para-ENUM-A is in CapIncFPT. This settles that OutputFPT = CapIncFPT and concludes the proof. □

The next theorem connects a collapse in the classical function world to a collapse in the parameterized function world. It witnesses the first connection between a collapse in classical and one in parameterized function complexity.

Theorem 4. TF(para-NP) = F(FPT) *if and only if* TFNP = FP.

Proof. Let us start with the easy direction.

"⇒": Let $A \subseteq \Sigma^* \times \Sigma^*$ be a total relation in TFNP. By definition of TFNP and TF(para-NP), $(A,\kappa_0) \in$ TF(para-NP), where κ_0 is the trivial parameterization assigning to each $x \in \Sigma^*$ the zero 0. Since TF(para-NP) = F(FPT), there exists a computable function $f\colon \mathbb{N} \to \mathbb{N}$, a polynomial p, and a deterministic algorithm $\mathcal{A}$, that, given the input $x \in \Sigma^*$, outputs some $y \in \Sigma^*$ such that $A(x,y)$ in time $f(\kappa_0(x)) \cdot p(x)$. As $\kappa_0(x) = 0$ for each x, $f(\kappa_0(x)) = f(0) \in \mathbb{N}$ for each x and $\mathcal{A}$ runs in polynomial time. Accordingly, we have $A \in$ FP and thereby FP = TFNP as A was chosen arbitrarily.

"⇐": Choose some $B \in$ TF(para-NP) via the nondeterministic machine M running in time $f(\kappa(x)) \cdot p(|x|)$ for a polynomial p, a computable function f, a parameterization κ, and an input x. By Proposition 1, we know that there exists a computable function $\pi\colon \Sigma^* \to \Sigma^*$ and a problem $B' \subseteq \Sigma^* \times \Sigma^* \times \Sigma^*$ such that $B' \in$ FNP and the following is true: for all instances $x \in \Sigma^*$ and all solutions $y \in \Sigma^*$, we have that $(x,y) \in B$ if and only if $\big((x,\pi(\kappa(x)),y\big) \in B'$.

As B is total, B' is total with respect to the third argument as well. It follows by assumption that B' is also in FP via some deterministic machine M' having a runtime bounded by a polynomial q in the input length. Accordingly, we can define a deterministic machine $\widetilde{M}$ for B which, given input $x \in \Sigma^*$, computes $\pi(\kappa(x))$, then simulates M' on $(x,\pi(\kappa(x)))$, and runs in time

$$f_\pi(\kappa(x)) + q(|\pi(\kappa(x))|,|x|), \tag{1}$$

where $f_\pi\colon \Sigma^* \to \mathbb{N}$ is a computable function that estimates the runtime of computing $\pi(\kappa(x))$. Clearly, Equation (1) is an fpt-runtime witnessing $B \in$ F(FPT). Accordingly, we can deduce that TF(para-NP) = F(FPT), as B was chosen arbitrarily. □

Combining Theorem 3 with Theorem 4 and finally Proposition 4 connects the parameterized enumeration world with the classical enumeration world. The following corollary summarizes deducible connections of parameterized enumeration to classical enumeration, parameterized function complexity, classical function complexity, and decision complexity. This means that a collapse of OutputFPT to

CapIncFPT would yield a collapse of TFNP to FP (Proposition 4) and as well as of P to NP ∩ coNP (due to TFNP = F(NP ∩ coNP) [23]).

Corollary 3. *The following collapses are equivalent:*

- OutputFPT = CapIncFPT
- OutputP = CapIncP
- TF(para-NP) = F(FPT)
- TFNP = FP
- P = NP ∩ coNP

If one does not consider space requirements—which is usually the case for these classes—we can deduce the following corollary by applying Corollary 2 and Proposition 2.

Corollary 4. *Without space requirements, we have that* OutputFPT = IncFPT *if and only if* OutputP = IncP.

Now, we have seen that the observations made by Capelli and Strozecki [12] can be directly transferred to the parameterized setting. In cryptography, one-way functions are functions that are easily (in polynomial time) to compute but hard to invert (often, this means either it requires exponential time to invert the function, or, in a probabilistic model, the search space is of exponential size wherefore the success probability to invert the function is very low). A good introduction to this topic is given by Talbot and Welsh [37]. As a result of Corollary 3 together with Corollary 4, for instance, the existence of one-way functions would separate OutputFPT from IncFPT as well.

5. Conclusions and Outlook

We presented the first connection of parameterized enumeration to classical enumeration 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 parameterized function classes developed in this paper to collapses of classical function classes. In particular, we proved that TF(para-NP) = F(FPT) if and only if TFNP = FP (Theorem 4). The function complexity class TFNP, which has TF(para-NP) as its parameterized counterpart, contains significant cryptography-related problems such as factoring. Furthermore, we studied a parameterized incremental FPT time enumeration hierarchy on the level of exponent slices (Definition 8) and observed that $\mathsf{CapIncFPT}_1$ = DelayFPT (Corollary 2). Additionally, an interleaving of the two hierarchies, IncFPT_a and $\mathsf{CapIncFPT}_a$, has been shown (Theorem 1). The results of this paper underline that parameterized enumeration complexity is an area worth studies, as there are deep connections to its classical counterpart.

Future research should build on these classes to unveil the presence of exponential space in this setting and give a definite answer to the observed time–space tradeoffs of IncFPT versus DelayFPT. Can the hierarchy IncFPT_a introduced and studied in Section 3, be shown strict under similar conditions as Capelli and Strozecki were able to prove (they used the exponential time hypothesis)? Moreover, it would be engaging to study connections from parameterized enumeration to proof theory via the work of Goldberg and Papadimitriou [24]. We want to close with the question of whether there exist intermediate natural problems between F(FPT) and TF(para-NP) which are relevant in some area beyond trivial parameterization $\kappa_{\text{one}}(x) = 1$.

Funding: This research was funded by Deutsche Forschungsgemeinschaft (ME 4279/1-2).

Acknowledgments: The author thanks Maurice Chandoo (Hannover), Johannes Fichte (Dresden), Martin Lück (Hannover), Johannes Schmidt (Jönköping), and Till Tantau (Lübeck) for various discussions about topics of the paper.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Johnson, D.S.; Papadimitriou, C.H.; Yannakakis, M. On Generating All Maximal Independent Sets. *Inf. Process. Lett.* **1988**, *27*, 119–123. [CrossRef]
2. Avis, D.; Fukuda, K. Reverse Search for Enumeration. *Discrete Appl. Math.* **1996**, *65*, 21–46. [CrossRef]
3. Creignou, N.; Olive, F.; Schmidt, J. Enumerating All Solutions of a Boolean CSP by Non-decreasing Weight. In *Theory and Applications of Satisfiability Testing, Proceedings of the 14th International Conference, SAT 2011, Ann Arbor, MI, USA, 19–22 June 2011*; Sakallah, K.A., Simon, L., Eds.; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2011; Volume 6695, pp. 120–133. [CrossRef]
4. Strozecki, Y. On Enumerating Monomials and Other Combinatorial Structures by Polynomial Interpolation. *Theory Comput. Syst.* **2013**, *53*, 532–568. [CrossRef]
5. Creignou, N.; Hébrard, J.J. On generating all solutions of generalized satisfiability problems. *Theor. Inform. Appl.* **1997**, *31*, 499–511. [CrossRef]
6. Durand, A.; Strozecki, Y. Enumeration Complexity of Logical Query Problems with Second-order Variables. In *Computer Science Logic, Proceedings of the 25th International Workshop/20th Annual Conference of the EACSL, CSL 2011, Bergen, Norway, 12–15 September 2011*; Bezem, M., Ed.; Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik: Wadern, Germany, 2011; Volume 12, pp. 189–202. [CrossRef]
7. Read, R.C.; Tarjan, R.E. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks* **1975**, *5*, 237–252. [CrossRef]
8. Khachiyan, L.G.; Boros, E.; Elbassioni, K.M.; Gurvich, V.; Makino, K. On the Complexity of Some Enumeration Problems for Matroids. *SIAM J. Discret. Math.* **2005**, *19*, 966–984. [CrossRef]
9. Eiter, T.; Gottlob, G.; Makino, K. New Results on Monotone Dualization and Generating Hypergraph Transversals. *SIAM J. Comput.* **2003**, *32*, 514–537. [CrossRef]
10. Fredman, M.L.; Khachiyan, L. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *J. Algorithms* **1996**, *21*, 618–628. [CrossRef]
11. Carmeli, N.; Kenig, B.; Kimelfeld, B. Efficiently Enumerating Minimal Triangulations. In Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2017), Chicago, IL, USA, 14–19 May 2017; Sallinger, E., den Bussche, J.V., Geerts, F., Eds.; ACM: New York, NY, USA, 2017; pp. 273–287. [CrossRef]
12. Capelli, F.; Strozecki, Y. Incremental delay enumeration: Space and time. *Discret. Appl. Math.* **2019**, *268*, 179–190. [CrossRef]
13. Downey, R.G.; Fellows, M.R. *Fundamentals of Parameterized Complexity*; Springer: London, UK, 2013. [CrossRef]
14. Downey, R.G.; Fellows, M.R. *Parameterized Complexity*; Springer: New York, NY, USA, 1999. [CrossRef]
15. Flum, J.; Grohe, M. *Parameterized Complexity Theory*; Texts in Theoretical Computer Science. An EATCS Series; Springer: Berlin, Germany, 2006. [CrossRef]
16. Niedermeier, R. *Invitation to Fixed-Parameter Algorithms*; Oxford University Press: Oxford, UK, 2006.
17. Alber, J.; Fernau, H.; Niedermeier, R. Parameterized complexity: Exponential speed-up for planar graph problems. *J. Algorithms* **2004**, *52*, 26–56. [CrossRef]
18. Creignou, N.; Meier, A.; Müller, J.; Schmidt, J.; Vollmer, H. Paradigms for Parameterized Enumeration. In *Mathematical Foundations of Computer Science 2013, Proceedings of the 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, 26–30 August 2013*; Chatterjee, K., Sgall, J., Eds.; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2013; Volume 8087, pp. 290–301. [CrossRef]
19. Creignou, N.; Meier, A.; Müller, J.; Schmidt, J.; Vollmer, H. Paradigms for Parameterized Enumeration. *Theory Comput. Syst.* **2017**, *60*, 737–758. [CrossRef]

20. Creignou, N.; Ktari, R.; Meier, A.; Müller, J.; Olive, F.; Vollmer, H. Parameterized Enumeration for Modification Problems. In *Language and Automata Theory and Applications, Proceedings of the 9th International Conference, LATA 2015, Nice, France, 2–6 March 2015*; Dediu, A., Formenti, E., Martín-Vide, C., Truthe, B., Eds.; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2015; Volume 8977, pp. 524–536. [CrossRef]
21. Creignou, N.; Ktari, R.; Meier, A.; Müller, J.; Olive, F.; Vollmer, H. Parameterised Enumeration for Modification Problems. *Algorithms* **2019**, *12*, 189. [CrossRef]
22. Meier, A.; Reinbold, C. Enumeration Complexity of Poor Man's Propositional Dependence Logic. In *Foundations of Information and Knowledge Systems, Proceedings of the 10thInternational Symposium, FoIKS 2018, Budapest, Hungary, 14–18 May 2018*; Ferrarotti, F., Woltran, S., Eds.; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2018.
23. Megiddo, N.; Papadimitriou, C.H. On Total Functions, Existence Theorems and Computational Complexity. *Theor. Comput. Sci.* **1991**, *81*, 317–324. [CrossRef]
24. Goldberg, P.W.; Papadimitriou, C.H. Towards a Unified Complexity Theory of Total Functions. *Electron. Colloq. Comput. Complex. (ECCC)* **2017**, *24*, 56. [CrossRef]
25. Chauhan, A.; Rao, B.V.R. Parameterized Analogues of Probabilistic Computation. In *Algorithms and Discrete Applied Mathematics, Proceedings of the First International Conference, CALDAM 2015, Kanpur, India, 8–10 February 2015*; Ganguly, S., Krishnamurti, R., Eds.; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2015; Volume 8959, pp. 181–192. [CrossRef]
26. Fichte, J.K.; Hecher, M.; Schindler, I. Default Logic and Bounded Treewidth. In *Language and Automata Theory and Applications Proceedings of the 12th International Conference, LATA 2018, Ramat Gan, Israel, 9–11 April 2018*; Klein, S.T., Martín-Vide, C., Shapira, D., Eds.; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2018; Volume 10792, pp. 130–142. [CrossRef]
27. Bläsius, T.; Friedrich, T.; Meeks, K.; Schirneck, M. On the Enumeration of Minimal Hitting Sets in Lexicographical Order. *arXiv* **2018**, arXiv:1805.01310.
28. Pichler, R.; Rümmele, S.; Woltran, S. Counting and Enumeration Problems with Bounded Treewidth. In *Logic for Programming, Artificial Intelligence, and Reasoning, Proceedings of the 16th International Conference, LPAR-16, Dakar, Senegal, 25 April–1 May 2010*; Clarke, E.M., Voronkov, A., Eds.; Revised Selected Papers; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2010; Volume 6355, pp. 387–404. [CrossRef]
29. Mary, A.; Strozecki, Y. Efficient Enumeration of Solutions Produced by Closure Operations. In Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016), Orléans, France, 17–20 February 2016; Ollinger, N., Vollmer, H., Eds.; Leibniz International Proceedings in Informatics (LIPIcs); Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2016; Volume 47, pp. 52:1–52:13. [CrossRef]
30. Flum, J.; Grohe, M. The Parameterized Complexity of Counting Problems. *SIAM J. Comput.* **2004**, *33*, 892–922. [CrossRef]
31. McCartin, C. Parameterized counting problems. *Ann. Pure Appl. Log.* **2006**, *138*, 147–182. [CrossRef]
32. Roth, M.; Wellnitz, P. Counting and Finding Homomorphisms is Universal for Parameterized Complexity Theory. In Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA 2020), Salt Lake City, UT, USA, 5–8 January 2020; Chawla, S., Ed.; SIAM: Philadelphia, PA, USA, 2020; pp. 2161–2180. [CrossRef]
33. Curticapean, R.; Dell, H.; Roth, M. Counting Edge-injective Homomorphisms and Matchings on Restricted Graph Classes. *Theory Comput. Syst.* **2019**, *63*, 987–1026. [CrossRef]
34. Roth, M.; Schmitt, J. Counting Induced Subgraphs: A Topological Approach to #W[1]-hardness. In Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC 2018), Helsinki, Finland, 20–24 August 2018; Paul, C., Pilipczuk, M., Eds.; Schloss Dagstuhl—Leibniz-Zentrum für Informatik: Wadern, Germany, 2018; Volume 115, pp. 24:1–24:14. [CrossRef]

35. Dörfler, J.; Roth, M.; Schmitt, J.; Wellnitz, P. Counting Induced Subgraphs: An Algebraic Approach to #W[1]-hardness. In Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019), Aachen, Germany, 26–30 August 2019; Rossmanith, P., Heggernes, P., Katoen, J., Eds.; Schloss Dagstuhl—Leibniz-Zentrum für Informatik: Wadern, Germany, 2019; Volume 138, pp. 26:1–26:14. [CrossRef]
36. Curticapean, R. The Simple, Little and Slow Things Count: On Parameterized Counting Complexity. Ph.D. Thesis, Saarland University, Saarbrücken, Germany, 2015.
37. Talbot, J.M.; Welsh, D.J.A. *Complexity and Cryptography—An Introduction*; Cambridge University Press: Cambridge, UK, 2006.
38. Creignou, N.; Kröll, M.; Pichler, R.; Skritek, S.; Vollmer, H. On the Complexity of Hard Enumeration Problems. In *Language and Automata Theory and Application, Proceedings of the 11th International Conference, LATA 2017, Umeå, Sweden, 6–9 March 2017*; Drewes, F., Martín-Vide, C., Truthe, B., Eds.; Springer: Cham, Switzerland, 2017; Volume 10168, pp. 183–195. [CrossRef]
39. Pippenger, N. *Theories of Computability*; Cambridge University Press: Cambridge, UK, 1997.
40. Schmidt, J. Enumeration: Algorithms and Complexity. Master's Thesis, Leibniz Universität Hannover, Hanover, Germany, 2009. Available online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.582.8008&rep=rep1&type=pdf (accessed on 14 May 2020).
41. Fomin, F.V.; Kratsch, D. *Exact Exponential Algorithms*; Springer: Berlin, Germany, 2010. [CrossRef]
42. Hartmanis, J.; Stearns, R.E. On the computational complexity of algorithms. *Trans. Am. Math. Soc.* **1965**, *117*, 285–306. [CrossRef]
43. Bellare, M.; Goldwasser, S. The Complexity of Decision Versus Search. *SIAM J. Comput.* **1994**, *23*, 97–119. [CrossRef]
44. Johnson, D.S.; Papadimitriou, C.H.; Yannakakis, M. How Easy is Local Search? *J. Comput. Syst. Sci.* **1988**, *37*, 79–100. [CrossRef]