Electronic Colloquium on Computational Complexity, Report No. 130 (2015)

ECCC

An Overview of Non-Uniform Parameterized Complexity

Ronald de Haan

Abstract

We consider several non-uniform variants of parameterized complexity classes that have been considered in the literature. We do so in a homogenous notation, allowing a clear comparison of the various variants. Additionally, we consider some novel (non-uniform) parameterized complexity classes that come up in the framework of parameterized knowledge compilation. We provide some (inclusion and separation) results relating the different non-uniform classes to each other. Moreover, we illustrate how these non-uniform parameterized complexity classes are useful in the setting of parameterized knowledge compilation.

Contents

1 Introduction **2**

2 Non-uniform parameterized complexity classes **2**
2.1 Fpt-size and xp-size advice . . . 3
2.2 Slice-wise advice . . . 3
2.3 Poly-size and kernel-size advice . . . 4
2.4 Slice-wise non-uniformity . . . 4

3 Basic results **6**
3.1 Alternative characterizations . . . 6
3.1.1 Characterizations in terms of fpt-reductions with advice . . . 6
3.1.2 Circuit characterizations . . . 7
3.2 Separations . . . 7

4 Relation to Parameterized Knowledge Compilation **12**
4.1 Parameterized Knowledge Compilation . . . 12
4.1.1 Compilability . . . 13
4.1.2 Parameterized Compilability . . . 13
4.2 (Conditional) incompilability results . . . 15
4.3 Restricting the instance space . . . 17
4.3.1 Normalization results . . . 19
4.3.2 Relating few-NP and nu-few-NP to other classes . . . 19
4.3.3 Complete problems . . . 20
4.3.4 Differences between non-uniform variants of few-NP . . . 21
4.3.5 Characterizing few-NP in terms of 3-colorability . . . 21
4.3.6 Characterizing few-NP by “filtering” para-NP problems . . . 22
4.4 The parameterized compilability of finding small cliques . . . 24
4.4.1 Clique size as part of the offline instance . . . 27
4.5 Other parameterized compilation problems . . . 27
4.5.1 Hamiltonian Paths and the Travelling Salesman Problem . . . 27
4.5.2 Graph Colorability . . . 28
4.5.3 Other Problems . . . 29

ISSN 1433-8092

5 Relation to (a Parameterized Variant of) the Polynomial Hierarchy **30**

6 Relation to Probabilistic Parameterized Complexity **34**

7 Conclusion **34**

1 Introduction

In various areas of parameterized complexity analysis, the notion of non-uniform computation shows up. In the literature, several non-uniform variants of parameterized complexity classes have been considered, e.g., see [5, 8, 14]. These variants are defined in very different ways, and are not (or hardly) related to each other.

We give an overview of different non-uniform variants of parameterized complexity classes. We do so in a homogeneous notation, that allows a clear comparison of the different variants. Moreover, we provide some (inclusion and separation) results relating the different non-uniform classes to each other.

Additionally, we illustrate how these non-uniform parameterized complexity classes are useful for various applications of the framework of parameterized complexity. Most notably, we discuss the relation between parameterized knowledge compilation and non-uniform parameterized complexity. We introduce several novel (non-uniform) parameterized complexity classes that are useful in the setting of parameterized knowledge compilation. We show how these classes can be used in this setting, and we relate them to the other non-uniform parameterized complexity classes that we consider.

2 Non-uniform parameterized complexity classes

In this section, we give a definition of the non-uniform parameterized complexity classes that we discuss in the paper. Most classes can be defined in a homogeneous way, namely by using advice. Only for the class XP_{nu}, a natural non-uniform variant of the commonly considered class XP, the most natural definition is of a different form. For the sake of completeness, we begin with a brief overview of relevant (uniform) parameterized complexity classes.

An overview of all (non-uniform) parameterized complexity classes considered in this paper, including the classes defined in this section can be found in Figure 1.

Preliminary definitions We briefly introduce some core notions from parameterized complexity theory. For an in-depth treatment we refer to other sources [8, 9, 14, 26]. A *parameterized problem* L is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. In the remainder of this paper, we fix an arbitrary alphabet Σ. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, we call x the *main part* and k the *parameter*. A parameterized problem L is *fixed-parameter tractable* if there exists a computable function f and a constant c such that there exists an algorithm that decides whether $(x, k) \in L$ in time $O(f(k)|x|^c)$, where $|x|$ denotes the size of x. We let FPT denote the class of all fixed-parameter tractable parameterized problems. We define XP to be the class of all parameterized problems L for which there exists a computable function f and an algorithm that decides whether $(x, k) \in L$ in time $|x|^{f(k)}$.

Let $L \subseteq \Sigma^* \times \mathbb{N}$ and $L' \subseteq (\Sigma')^* \times \mathbb{N}$ be two parameterized problems. An *fpt-reduction* from L to L' is a mapping $R : \Sigma^* \times \mathbb{N} \to (\Sigma')^* \times \mathbb{N}$ from instances of L to instances of L' such that there exist some computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $(I, k) \in \Sigma^* \times \mathbb{N}$: (i) (I, k) is a yes-instance of L if and only if $(I', k') = R(I, k)$ is a yes-instance of L', (ii) $k' \leq g(k)$, and (iii) R is computable in fpt-time. Similarly, we call reductions that satisfy properties (i) and (ii) but that are computable in time $|x|^{f(k)}$, for some fixed computable function f, *xp-reductions*.

When $\mathcal{S}$ is a set of parameterized problems, we let $[\mathcal{S}]_{\text{fpt}}$ denote the closure of S under fpt-reductions, i.e., $[\mathcal{S}]_{\text{fpt}}$ consists of all parameterized problems that can be fpt-reduced to some problem $S \in \mathcal{S}$.

For any parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ and any positive integer $k \in \mathbb{N}$, we call the unparameterized problem $L_k = \{x : (x, k) \in L\}$ the *k-th slice of L*.

Let C be a classical complexity class, e.g., NP. The parameterized complexity class para-C is then defined as the class of all parameterized problems $L \subseteq \Sigma^* \times \mathbb{N}$, for some finite alphabet Σ, for which there exist an alphabet Π, a computable function $f : \mathbb{N} \to \Pi^*$, and a problem $P \subseteq \Sigma^* \times \Pi^*$ such that $P \in C$ and for all instances $(x, k) \in \Sigma^* \times \mathbb{N}$

of L we have that $(x,k) \in L$ if and only if $(x, f(k)) \in P$. Intuitively, the class para-C consists of all problems that are in C after a precomputation that only involves the parameter [13]. The class para-NP can also be defined as the class of all parameterized problems that can be decided in nondeterministic fixed-parameter tractable time.

The parameterized complexity classes W[t], t ≥ 1, W[SAT] and W[P] are based on the satisfiability problems of Boolean circuits and formulas. We consider *Boolean circuits* with a single output gate. We call input nodes *variables*. We distinguish between *small* gates, with fan-in ≤ 2, and *large gates*, with fan-in > 2. The *depth* of a circuit is the length of a longest path from any variable to the output gate. The *weft* of a circuit is the largest number of large gates on any path from a variable to the output gate. A *Boolean formula* can be considered as a Boolean circuit where all gates have fan-out ≤ 1. We adopt the usual notions of truth assignments and satisfiability of a Boolean circuit. We say that a truth assignment for a Boolean circuit has *weight* k if it sets exactly k of the variables of the circuit to true. We denote the class of Boolean circuits with depth u and weft t by $\text{CIRC}_{t,u}$. We denote the class of all Boolean circuits by CIRC, and the class of all Boolean formulas by FORM. For any class $\mathcal{C}$ of Boolean circuits, we define the following parameterized problem.

$\text{WSAT}(\mathcal{C})$
Instance: A Boolean circuit $C \in \mathcal{C}$, and an integer k.
Parameter: k.
Question: Does there exist an assignment of weight k that satisfies C?

The classes W[t] for t ≥ 1 and the classes W[SAT] and W[P] are defined as follows:

$$\begin{aligned} \text{W[t]} &= [\,\{\,\text{WSAT}(\text{CIRC}_{t,u}) : u \geq 1\,\}\,]_{\text{fpt}}, \text{ for each t} \geq 1; \\ \text{W[SAT]} &= [\,\text{WSAT}(\text{FORM})\,]_{\text{fpt}}; \text{ and} \\ \text{W[P]} &= [\,\text{WSAT}(\text{CIRC})\,]_{\text{fpt}}. \end{aligned}$$

2.1 Fpt-size and xp-size advice

We begin our exposition of non-uniform parameterized complexity classes with several classes that are based on advice depending on the input size n and the parameter value k. Such classes, where the advice string α is of fpt-size, have been considered in the context of parameterized knowledge compilation [5]. We define these classes, as well as a natural variant where xp-size advice is allowed.

Definition 1 (*fpt-size advice*)**.** *Let C be a parameterized complexity class. We define C/fpt to be the class of all parameterized problems Q for which there exists a parameterized problem $Q' \in C$, a computable function f and a constant c such that for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there exists some $\alpha(n,k) \in \Sigma^*$ of size $f(k)n^c$ with the property that for all instances (x,k) with $|x| = n$, it holds that $(x,k) \in Q$ if and only if $(x, \alpha(|x|,k), k) \in Q'$.*

The classes C/fpt have been defined by Chen as C/ppoly [5].

Definition 2 (*xp-size advice*)**.** *Let C be a parameterized complexity class. We define C/xp to be the class of all parameterized problems Q for which there exists a parameterized problem $Q' \in C$ and a computable function f such that for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there exists some $\alpha(n,k) \in \Sigma^*$ of size $n^{f(k)}$ with the property that for all instances (x,k) with $|x| = n$, it holds that $(x,k) \in Q$ if and only if $(x, \alpha(|x|,k), k) \in Q'$.*

The above definitions have the following direct consequence.

Observation 3. *It holds that* $\text{FPT}/\text{xp} = \text{XP}/\text{xp}$. *We will use the notation* XP/xp *for this class to emphasize that an $n^{f(k)}$ running time is allowed for algorithms that witness membership in this class.*

2.2 Slice-wise advice

We continue our exposition with non-uniform variants of parameterized complexity classes that get advice (of computable size) for each slice.

Definition 4 (*slice-wise advice*)**.** *Let C be a parameterized complexity class. We define $C/$*slice *to be the class of all parameterized problems Q for which there exists a parameterized problem $Q' \in C$ and a computable function f such that for each $k \in \Sigma^*$ there exists some advice string $\alpha(k) \in \Sigma^*$ of size at most $f(k)$ with the property that for all instances (x, k) with $|x| = n$, it holds that $(x, k) \in Q$ if and only if $(x, \alpha(k), k) \in Q'$.*

Intuitively, the difference between the definitions of the classes $C/$slice and $C/$xp can be explained as follows by taking as example $C = $ XP. For problems in the class XP$/$slice, for each parameter value k there must be a single (uniformly defined) polynomial-time algorithm (whose descriptions must be of size computable in k). For problems in the class XP$/$xp, for each parameter value k there can be a (non-uniformly defined) polynomial-time algorithm (with similar size bounds).

Observation 5. *The class* FPT$/$slice *can straightforwardly be shown to be equivalent to* P$/$slice*. Here we consider* P *as a parameterized complexity class.*

Observation 6. *For each parameterized complexity class C, it holds that $C \subseteq C/$slice $\subseteq C/$fpt $\subseteq C/$xp.

2.3 Poly-size and kernel-size advice

Next, we present some additional natural non-uniform variants of parameterized complexity classes that are also based on advice. These variants are natural notions of non-uniform complexity that come up in analogy to the classes defined in Section 2.1.

Definition 7 (*poly-size advice*)**.** *Let C be a parameterized complexity class. We define $C/$*poly *to be the class of all parameterized problems Q for which there exists a parameterized problem $Q' \in C$ and a constant c such that for each $(n, k) \in \mathbb{N} \times \Sigma^*$ there exists some $\alpha(n, k) \in \Sigma^*$ of size n^c with the property that for all instances (x, k) with $|x| = n$, it holds that $(x, k) \in Q$ if and only if $(x, \alpha(|x|, k), k) \in Q'$.*

Observation 8. *For each parameterized complexity class C, we have that $C \subseteq C/$poly $\subseteq C/$fpt.

Definition 9 (*kernel-size advice*)**.** *Let C be a parameterized complexity class. We define $C/$*kernel *to be the class of all parameterized problems Q for which there exists a parameterized problem $Q' \in C$ and a computable function f such that for each $(n, k) \in \mathbb{N} \times \Sigma^*$ there exists some $\alpha(n, k) \in \Sigma^*$ of size $f(k)$ with the property that for all instances (x, k) with $|x| = n$, it holds that $(x, k) \in Q$ if and only if $(x, \alpha(|x|, k), k) \in Q'$.*

Observation 10. *For each parameterized complexity class C, we have that $C \subseteq C/$kernel $\subseteq C/$fpt.

2.4 Slice-wise non-uniformity

The last variant of non-uniformity that we consider can be called slice-wise non-uniformity. Firstly, we consider the class FPT_{nu}, where problems are required to be solvable in fpt-time, but the algorithm for each slice can be different.

Definition 11 ([8])**.** *The parameterized complexity class* FPT_{nu} *is defined as the class of all parameterized problems Q for which there exists a (possibly uncomputable) function f and a constant c such that for every $k \in \mathbb{N}$, the k-th slice Q_k of Q is decidable in time $f(k)n^c$, where n is the size of the instance.*

Note that the class FPT_{nu} is the class of parameterized problems that are *non-uniformly fixed-parameter tractable* in the sense that is discussed in textbooks [8, 14]. Moreover, it coincides with a variant of the class FPT$/$slice, where the function f bounding the size of the advice $\alpha(k)$ is not required to be computable (see [4, Theorem 1.3]).

Finally, we give a definition of the most prominent non-uniform variant of XP (that is also presented in textbooks on parameterized complexity [8, 14]).

Definition 12 ([8])**.** *The parameterized complexity class* XP_{nu} *is defined as the class of all parameterized problems Q for which for each $k \in \Sigma^*$, the slice $Q_k = \{ x : (x, k) \in Q \}$ is polynomial-time solvable.*

We point out that in the definition of the class XP_{nu}, the order of the polynomial that is a bound on the running time of the algorithm that solves slice Q_k of a problem $Q \in \text{XP}_{\text{nu}}$ is allowed to vary for different values of k, but it does not have to be bounded by a computable function of k. In contrast, for problems Q in FPT_{nu}, each slice Q_k must be solvable in time $O(n^c)$ for some fixed constant c; here the factor $f(k)$ hidden by the big-oh notation does in general depend on k.

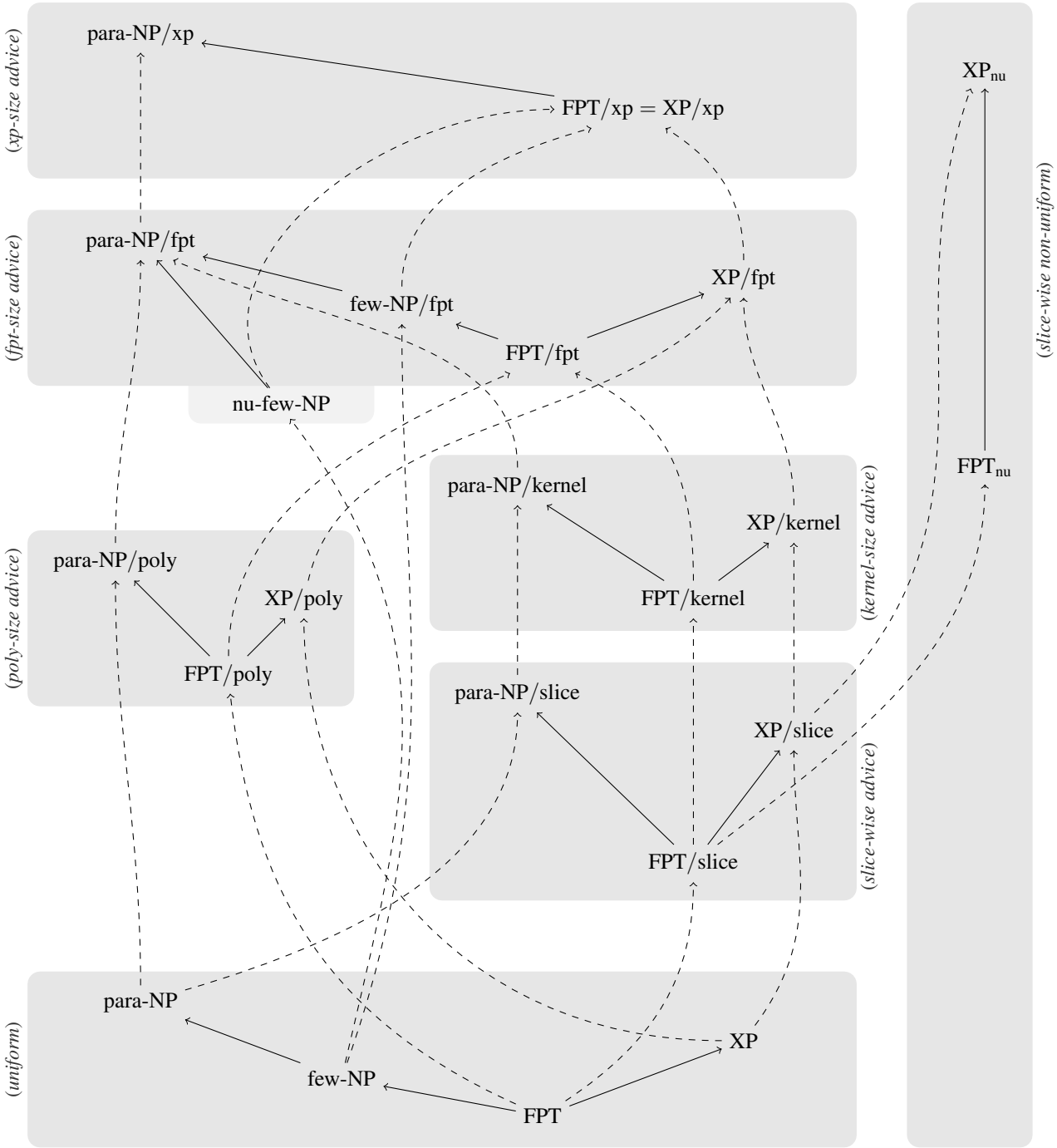

Figure 1: Overview of (non-uniform) parameterized complexity classes. Most classes are defined in Section 2. For a definition of few-NP, nu-few-NP and few-NP/fpt, we refer to Section 4.3.

3 Basic results

In this section, we provide some basic results about the different non-uniform parameterized complexity classes (that we defined in the previous section). We begin with giving several alternative characterizations of some of the classes. Then we relate several of the classes to each other, by giving some separation (non-inclusion) results. A graphical overview of the relations between the classes can be found in Figure 1.

3.1 Alternative characterizations

We give alternative characterizations of the non-uniform complexity classes defined in Section 2, in terms of fpt-reductions with advice, in terms of Boolean circuits, and by using slice-wise solvability.

3.1.1 Characterizations in terms of fpt-reductions with advice

We show that the classes C/fpt, C/kernel and C/poly can be defined by means of fpt-reductions with advice of appropriate size. We start with defining fpt-reductions with fpt-size advice, and showing that C/fpt can be characterized using such reductions.

Definition 13 (fpt-reductions with fpt-size advice)**.** *Let Q, Q' be parameterized problems. We say that an fpt-algorithm R is* an fpt-reduction with fpt-size advice from Q to Q' *if there exist computable functions f, h and a constant c such that for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there is an advice string $\alpha(n,k)$ of length $f(k)n^c$ such that for each instance $(x,k) \in \Sigma^* \times \mathbb{N}$ with $|x| = n$, it holds that (1) $(x',k') = R(x,\alpha(n,k),k) \in Q'$ if and only if $(x,k) \in Q$, and (2) $k' \leq h(k)$.*

When there exists an fpt-reduction with fpt-advice from Q to Q', we say that Q is fpt-reducible to Q' with fpt-size advice.

Proposition 14. *Let C be a parameterized complexity class that is closed under fpt-reductions. Then C/fpt coincides with the class of parameterized problems that are fpt-reducible with fpt-size advice to some problem in C.*

Proof. ($\Rightarrow$) Take an arbitrary problem $Q \in C/\text{fpt}$. Then, by definition, there exists some $Q' \in C$ such that for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there exists some advice string $\alpha(n,k)$ of fpt-size with the property that for each $(x,k) \in \Sigma^* \times \Sigma^*$ with $|x| = n$ it holds that $(x,k) \in Q$ if and only if $(x,\alpha(n,k),k) \in Q'$. We construct an fpt-reduction R with fpt-size advice from Q to Q'. For each (n,k), we let the advice string for R be the string $\alpha(n,k)$. Moreover, we let $R(x,k) = (x,\alpha(n,k),k)$. This proves that Q is fpt-reducible to some problem in C with fpt-size advice.

($\Leftarrow$) Conversely, take an arbitrary parameterized problem Q that is fpt-reducible to some problem $Q' \in C$ with fpt-size advice, by an fpt-reduction R. By definition, then, for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there exists some fpt-size advice string $\alpha(n,k)$ with the property that for each $(x,k) \in \Sigma^* \times \Sigma^*$ with $|x| = n$, it holds that $R(x,\alpha(n,k),k) = (x',k') \in Q'$ if and only if $(x,k) \in Q$. We show that $Q \in C/\text{fpt}$ by specifying fpt-size advice for each $(n,k) \in \mathbb{N} \times \Sigma^*$ and giving a problem $Q'' \in C$ such that for each $(x,k) \in \Sigma^* \times \Sigma^*$ with $|x| = n$ it holds that $(x,k) \in Q$ if and only if $(x,\alpha(n,k),k) \in Q''$. For each $(n,k) \in \mathbb{N} \times \Sigma^*$ we let the advice string be $\alpha(n,k)$. Moreover, we let $Q'' = \{\, (x,y,k) \in \Sigma^* \times \Sigma^* \times \Sigma^* : R(x,y,k) \in Q' \,\}$. Since C is closed under fpt-reductions, we know that $Q'' \in C$. Therefore, $Q \in C/\text{fpt}$. □

Next, we extend this characterization by introducting fpt-reductions with kernel-size and polynomial-size advice, and showing that these can be used to characterize C/kernel and C/poly, respectively.

Definition 15 (fpt-reductions with kernel-size advice)**.** *Let Q, Q' be parameterized problems. We say that an fpt-algorithm R is* an fpt-reduction with kernel-size advice from Q to Q' *if there exist computable functions f, h such that for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there is an advice string $\alpha(n,k)$ of length $f(k)$ such that for each instance $(x,k) \in \Sigma^* \times \mathbb{N}$ with $|x| = n$, it holds that (1) $(x',k') = R(x,\alpha(n,k),k) \in Q'$ if and only if $(x,k) \in Q$, and (2) $k' \leq h(k)$.*

Proposition 16. *Let C be a parameterized complexity class that is closed under fpt-reductions. Then C/kernel coincides with the class of parameterized problems that are fpt-reducible with kernel-size advice to some problem in C.*

Proof. The proof of Proposition 14 can straightforwardly be modified to show this result. □

Definition 17 (fpt-reductions with poly-size advice)**.** *Let Q, Q' be parameterized problems. We say that an fpt-algorithm R is* an fpt-reduction with poly-size advice from Q to Q' *if there exist a computable function h and a constant c such that for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there is an advice string $\alpha(n,k)$ of length n^c such that for each instance $(x,k) \in \Sigma^* \times \mathbb{N}$ with $|x| = n$, it holds that (1) $(x',k') = R(x,\alpha(n,k),k) \in Q'$ if and only if $(x,k) \in Q$, and (2) $k' \leq h(k)$.*

Proposition 18. *Let C be a parameterized complexity class that is closed under fpt-reductions. Then C/poly coincides with the class of parameterized problems that are fpt-reducible with poly-size advice to some problem in C.*

Proof. The proof of Proposition 14 can straightforwardly be modified to show this result. □

3.1.2 Circuit characterizations

We provide the following alternative characterizations of the classes FPT/fpt and para-NP/fpt, in terms of (families of) circuits.

Let $\mathcal{C}$ be a circuit with m input nodes, and let $x \in \{0,1\}^*$ be a bitstring of length $n \leq m$. Then, by $\mathcal{C}[x]$ we denote the circuit obtained from $\mathcal{C}$ by instantiating the first n input nodes according to the n values in x.

Proposition 19. *The class* FPT/fpt *coincides with the set of all parameterized problems Q for which there exists a computable function f and a constant c such that for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there is some circuit $\mathcal{C}_{n,k}$ of size $f(k)n^c$ that decides, for each input $x \in \{0,1\}^n$ of length n, whether $(x,k) \in Q$.*

Proof (sketch). Let Q be a problem that is solvable in fpt-time using an fpt-size advice string $\alpha(n,k)$ that only depends on the input size n and the parameter value k. Then for any (n,k), one can "hard-code" this advice and its use by the algorithm in an fpt-size circuit $\mathcal{C}_{n,k}$ that decides for inputs x of length n whether $(x,k) \in Q$.

Conversely, if for a problem Q, for each (n,k) there exists an fpt-size circuit $\mathcal{C}_{n,k}$ that decides for inputs x of length n whether $(x,k) \in Q$, then one can decide Q in fpt-time by taking a description of this circuit as the advice string $\alpha(n,k)$, and simulating the circuit on any input x of length n. □

Proposition 20. *The class* para-NP/fpt *coincides with the set of all parameterized problems Q for which there exists a computable function f and a constant c such that for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there is some circuit $\mathcal{C}_{n,k}$ of size $f(k)n^c$ such that, for each input $x \in \{0,1\}^n$ of length n, it holds that $x \in Q$ if and only if $\mathcal{C}_{n,k}[x]$ is satisfiable.*

Proof (idea). An argument similar to the one in the proof of Proposition 19 can be used to show this result. In addition, one needs to use the well-known correspondence between nondeterministic algorithms and satisfiability of circuits, as used in the Cook-Levin Theorem [7, 20]. □

3.2 Separations

Next, we give some (conditional) non-inclusion results between the various non-uniform complexity classes that we considered. Some of these results can be shown rather straightforwardly, whereas others need a bit more technical machinery.

The class XP_{nu} clearly contains the class XP/slice, and its subclasses. We show that this containment is strict.

Proposition 21. *There exists a parameterized problem $Q \in (\text{XP}_{\text{nu}}) \setminus (\text{XP/slice})$.*

Proof. Let g be an increasing function that grows faster asymptotically than every computable function. For instance, we can let g be the busy-beaver function, where for each $n \in \mathbb{N}$ the value $g(n)$ is the maximum number of steps performed by any Turing machine with n states that halts on the empty string, when given the empty string as input. By the Time Hierarchy Theorem [18], we know that for each $m \in \mathbb{N}$ there exists some problem P_m that is solvable in time $O(n^m)$ but not in time $O(n^{m-1})$, where n denotes the input size. We then let Q be the following parameterized problem:

$$Q = \{\, (x,k) : k \in \mathbb{N}, x \in P_{g(k)} \,\}.$$

For each value k, we know by assumption that $P_{g(k)}$ is solvable in polynomial-time, namely in time $O(n^{g(k)})$. Therefore, $Q \in \text{XP}_{\text{nu}}$.

We claim that $Q \notin \text{XP/slice}$. We proceed indirectly, and suppose that $Q \in \text{XP/slice}$. Then there exist a computable function f such that for each $k \in \mathbb{N}$, there exists some advice string $\alpha(k)$ such that the problem of deciding if $(x, k) \in Q$, given $(x, \alpha(k), k)$, is solvable in time $n^{f(k)}$, where $n = |x|$ denotes the input size. We know that g grows faster asymptotically than f. Therefore, there exists some ℓ such that $g(\ell) > f(\ell)$. Consider the slice Q_ℓ of Q. By "hard-coding" the advice string $\alpha(\ell)$ in an algorithm A_ℓ, we can then solve Q_ℓ in time $n^{f(\ell)}$. However, by construction of Q, the slice Q_ℓ is not solvable in time $O(n^{g(\ell)-1})$, and thus also not solvable in time $O(n^{f(\ell)})$, which is a contradiction. Therefore, we can conclude that $Q \notin \text{XP/slice}$. □

Corollary 22. $\text{XP/slice} \subsetneq \text{XP}_{\text{nu}}$.

Next, we will show that the different notions of non-uniformity that are used in classes like FPT/fpt, on the one hand, and the class XP_{nu}, on the other hand, are incomparable. In order to show this, we begin by exhibiting a parameterized problem that is in FPT/fpt (and that is in fact also contained in FPT/kernel and FPT/poly) but that is not in XP_{nu}.

Proposition 23. *There exists a parameterized problem* $Q \in (\text{FPT/fpt}) \setminus (\text{XP}_{\text{nu}})$.

Proof. Let S be an undecidable unary set, e.g., the set of all strings 1^m such that the m-th Turing machine (in some enumeration of all Turing machines) halts on the empty input. We define the following parameterized problem Q:

$$Q = \{\, (s, 1) : s \in S \,\}.$$

Clearly, $Q \in \text{FPT/fpt}$, since for input size n there is at most one string s of length n such that $(s, 1) \in Q$. We claim that $Q \notin \text{XP}_{\text{nu}}$. We proceed indirectly, and we suppose that $Q \in \text{XP}_{\text{nu}}$. Then there is an algorithm A that decides the slice $Q_1 = \{\, x : (x, 1) \in Q \,\}$ in polynomial-time. However, then the algorithm A can straightforwardly be modified to decide the undecidable set S, which is a contradiction. Therefore, $Q \notin \text{XP}_{\text{nu}}$. □

Corollary 24. *There exists a parameterized problem* $Q \in (\text{FPT/poly}) \setminus (\text{XP}_{\text{nu}})$ *and there exists a parameterized problem* $Q \in (\text{FPT/kernel}) \setminus (\text{XP}_{\text{nu}})$.

Proof (sketch). The problem Q constructed in the proof of Proposition 23 is in fact contained both in (FPT/poly) and in (FPT/kernel), and not in XP_{nu}. □

In order to complete our incomparability result, we show that there exists also a problem that is contained in XP_{nu}, but that is not contained in XP/xp. This result can also be seen as a strenghtened version of Proposition 21.

Proposition 25. *There exists a parameterized problem* $Q \in (\text{XP}_{\text{nu}}) \setminus (\text{XP/xp})$.

Proof. In order to show this, we will use a few theoretical tools. Firstly, we will consider the busy-beaver function $g : \mathbb{N} \to \mathbb{N}$, where $g(n)$ is defined to be the maximum number of steps performed by any Turing machine with n states that halts on the empty string. It is well-known and it can be shown straightforwardly that g grows faster asymptotically than any computable function f.

Next, we will make use of the following facts. For each $n \in \mathbb{N}$, there are 2^{2^n} possible Boolean functions $F : \{0,1\}^n \to \{0,1\}$. Each such function is computed by a circuit of size at most $c2^n$, for some fixed constant c. (By the size of a circuit, we denote the number of bits required in a binary representation of the circuit.) Moreover, there are at most $2^{f(n)}$ circuits with n input nodes of size $f(n)$. Additionally, we can enumerate all circuits $\mathcal{C}_1^n, \mathcal{C}_2^n, \ldots$ with n input nodes (possibly allowing repetitions) in such a way that:

- for each $m \in \mathbb{N}$, computing the m-th circuit $\mathcal{C}_m^n$ in this enumeration can be done in time $O(m \cdot |\mathcal{C}_m^n|^2)$; and
- for each $\ell, \ell' \in \mathbb{N}$ such that $\ell < \ell'$ it holds that all circuits of size ℓ come before all circuits of size ℓ' in this enumeration.

With these theoretical tools in place, we can begin constructing the problem Q. Since we want $Q \in \text{XP}_{\text{nu}}$, we need to ensure that all slices Q_k of Q are solvable in polynomial-time. However, the order of the polynomials that bounds the running time of the algorithms solving the slices Q_k does not need to be bounded by a computable function. This is exactly the property that we will exploit. We will construct Q in such a way that the running time needed to solve the slices Q_k grows so fast that any XP/xp algorithm will make a mistake on some slice Q_k.

In addition to g, we will consider two other functions h_1, h_2, that are defined as follows. Consider the following (infinite) sequences:

$$h_1 = (\overbrace{g(1)^2, \ldots, g(1)^2}^{g(1) \text{ times}}, \overbrace{g(2)^2, \ldots, g(2)^2}^{g(2) \text{ times}}, \overbrace{g(3)^2, \ldots, g(3)^2}^{g(3) \text{ times}}, \ldots), \text{ and}$$

$$h_2 = (1, 2, \ldots, g(1), 1, 2, \ldots, g(2), 1, 2, \ldots, g(3), 1, 2, \ldots).$$

We define $h_1(x)$ to be the x-th element in the sequence h_1 and $h_2(x)$ to be the x-th element in the sequence h_2.

We then define $Q \subseteq \{0,1\}^* \times \mathbb{N}$ to be the parameterized problem that is recognized by the following algorithm A. We make sure that A runs in polynomial time on each slice Q_k. On input $(x,k) \in \{0,1\}^* \times \mathbb{N}$, where $|x| = n$, the algorithm A uses the enumeration $\mathcal{C}^n_1, \mathcal{C}^n_2, \ldots$ of circuits on n input nodes, and tries to compute the $h_2(k)$-th circuit $\mathcal{C}^n_{h_2(k)}$. If this takes more than $h_1(k)$ steps, the algorithm accepts (x,k); otherwise, the algorithm simulates $\mathcal{C}^n_{h_2(k)}$ on the input $x \in \{0,1\}^n$, and accepts if and only if x satisfies $\mathcal{C}^n_{h_2(k)}$. The algorithm A runs in time $O(h_1(k)^2 \cdot n)$, so for each $k \in \mathbb{N}$, it decides Q_k in linear time.

We now show that $Q \notin \text{XP/xp}$. We proceed indirectly, and suppose that $Q \in \text{XP/xp}$. Then there exists some computable function $f : \mathbb{N} \to \mathbb{N}$ such that for each $(n,k) \in \mathbb{N} \times \mathbb{N}$ there exists a circuit $\mathcal{C}_{n,k}$ of size $n^{f(k)}$ that decides for inputs $x \in \{0,1\}^n$ whether $(x,k) \in Q$. We now identify a pair $(n_0, k_0) \in \mathbb{N} \times \mathbb{N}$ of values such that $(n_0)^{f(k_0)} < 2^{2^{n_0}}$ and $g(k_0) \geq c2^{n_0}$. In other words, we want (n_0, k_0) to satisfy the property that $f(k_0) < 2^{n_0}/\log n_0$ and $c2^{n_0} \leq g(k_0)$. We can do this as follows. Since $g(k)$ grows faster asymptotically than any computable function, we know that there exists some k_0 such that $g(k_0) > c2^{f(k_0)}$. Moreover, we let $n_0 = f(k_0)$. We then get that $g(k_0) > c2^{n_0}$ and $f(k_0) = n_0 < 2^{n_0}/\log n_0$. Now, let $k_1 \in \mathbb{N}$ be the least integer such that $h_1(k_1) = g(k_0)$. Then, by construction of h_1, for each $0 \leq \ell < g(k_0)$, it holds that $h_1(k_1 + \ell) = g(k_0)$.

Now, since we know that $(n_0)^{f(k_0)} < 2^{2^{n_0}}$, we know that there exists some Boolean function $F_0 : \{0,1\}^{n_0} \to \{0,1\}$ on n_0 variables that is not computed by any circuit of size $(n_0)^{f(k_0)}$. However, this function F_0 can be computed by some circuit of size $c2^{n_0}$. Therefore, because $g(k_0) > c2^{n_0}$, there is some $0 \leq \ell < g(k_0)$ such that the circuit $\mathcal{C}^{n_0}_{h_2(k_1+\ell)}$ computes F_0. Moreover, we can choose ℓ in such a way that $\mathcal{C}^{n_0}_{h_2(k_1+\ell)}$ is of size at most $c2^{n_0} < g(k_0)$. Therefore, computing $\mathcal{C}^{n_0}_{h_2(k_1+\ell)}$, using the enumeration $\mathcal{C}^{n_0}_1, \mathcal{C}^{n_0}_2, \ldots$, can be done in time $h_2(k_1+\ell)^2 \leq h_1(k_1+\ell) = g(k_0)$.

Now consider the pair $(n_0, k_0) \in \mathbb{N} \times \mathbb{N}$ of values. We assumed that there is some circuit $\mathcal{C}_{n_0,k_0}$ of size $(n_0)^{f(k_0)}$ that decides for inputs $x \in \{0,1\}^{n_0}$ whether $(x, k_0) \in Q$. By construction of Q and by the choice of (n_0, k_0), we know that for each $x \in \{0,1\}^{n_0}$ it must hold that $(x, k_0) \in Q$ if and only if $F_0(x) = 1$. This means that the circuit $\mathcal{C}_{n_0,k_0}$ computes F_0. However, since $\mathcal{C}_{n_0,k_0}$ is of size $(n_0)^{f(k_0)}$, this is a contradiction with the fact that F_0 is not computable by a circuit of size $(n_0)^{f(k_0)}$. Therefore, we can conclude that there exists no family of circuits $\mathcal{C}_{n,k}$ of size $n^{f(k)}$ for $(n,k) \in \mathbb{N} \times \mathbb{N}$. In other words, $Q \notin \text{XP/xp}$. □

The above results (Propositions 23 and 25 and Corollary 24) give us the incomparability results that we were after, which can be summarized as follows.

Corollary 26. *Let C be a parameterized complexity class such that* FPT/poly $\subseteq C \subseteq$ XP/xp. *Then C and* XP_{nu} *are incomparable w.r.t. set-inclusion, i.e., $C \not\subseteq \text{XP}_{\text{nu}}$ and $\text{XP}_{\text{nu}} \not\subseteq C$.*

Corollary 27. *Let C be a parameterized complexity class such that* FPT/kernel $\subseteq C \subseteq$ XP/xp. *Then C and* XP_{nu} *are incomparable w.r.t. set-inclusion, i.e., $C \not\subseteq \text{XP}_{\text{nu}}$ and $\text{XP}_{\text{nu}} \not\subseteq C$.*

In fact, the proof of Proposition 25 shows that there is a problem Q that is in FPT_{nu} but not in XP/xp. This also gives us the following results.

Corollary 28. *There exists a problem $Q \in (\text{FPT}_{\text{nu}}) \backslash (\text{XP/xp})$.*

Proof. The problem Q constructed in the proof of Proposition 25 is in $(\text{FPT}_{\text{nu}})\backslash(\text{XP}/\text{xp})$. □

Corollary 29. *Let C be a parameterized complexity class such that either* $\text{FPT}/\text{poly} \subseteq C \subseteq \text{XP}/\text{xp}$ *or* $\text{FPT}/\text{kernel} \subseteq C \subseteq \text{XP}/\text{xp}$. *Then C and* FPT_{nu} *are incomparable w.r.t. set-inclusion, i.e., $C \not\subseteq \text{FPT}_{\text{nu}}$ and $\text{FPT}_{\text{nu}} \not\subseteq C$.*

Next, we explore the power of the non-uniformity resulting from kernel-size and polynomial-size advice. Firstly, we give an easy proof that both forms of non-uniformity are more powerful than the uniform setting.

Proposition 30. *For every (nontrivial) decidable parameterized complexity class C, there is a parameterized problem $Q \in (C/\text{poly}) \setminus C$ and a parameterized problem $Q' \in (C/\text{kernel}) \setminus C$.*

Proof. The problem Q as constructed in the proof of Proposition 23 is also contained in C/poly and C/kernel, for each nontrivial C. However, since Q is undecidable, we know that $Q \notin C$. □

Secondly, we show that the non-uniform parameterized complexity classes FPT/kernel and FPT/poly are incomparable.

Proposition 31. *There is a parameterized problem $Q \in (\text{FPT}/\text{poly}) \setminus (\text{FPT}/\text{kernel})$.*

Proof. To show this result, it suffices to give a (classical) problem Q that is in P/poly but that is not solvable in polynomial time using constant-size advice. If we have such a problem Q, then the parameterized problem $\{ (x, 1) : x \in Q \}$ is in $(\text{FPT}/\text{poly}) \setminus (\text{FPT}/\text{kernel})$.

We use a diagonalization argument to construct the problem Q. Let $(A_1, c_1), (A_2, c_2), \ldots$ be an enumeration of all pairs (A_i, c_i) consisting of a polynomial-time algorithm A_i and a constant c_i. We will construct Q in such a way that for each algorithm A_i and each advice string y of length c_i, the algorithm A_i, when using the advice string y, makes a mistake on some input. We construct Q in stages: one stage for each (A_i, c_i). In stage i, we make sure that A_i makes a mistake for each advice string y of length c_i.

We now describe how to construct Q in a given stage i. Let (A_i, c_i) be the pair consisting of a polynomial-time algorithm A_i and a constant c_i. Take an input size n that has not yet been considered in previous stages, and such that $n \geq 2^{c_i}$. Since we have infinitely many input sizes at our disposal, we can always find such an n. Next, consider $2^{c_i} = u$ many (arbitrary, but different) input strings $x_1, \ldots, x_u$ of length n. Moreover, consider all possible advice strings $y_1, \ldots, y_u$ of length c_i. For each $1 \leq j \leq u$, we let $x_j \in Q$ if and only if the algorithm A_i, using the advice string y_j, rejects the input string x_j. For all other input strings x of length n, we let $x \notin Q$. Note that Q contains at most $2^{c_i} \leq n$ many strings of length n. This completes stage i. For all input lengths n that are not considered in any stage of our construction, we let Q contain no strings of length n.

We show that $Q \in \text{P}/\text{poly}$. We define the advice for input size n to be a table consisting of all strings in Q of length n. Since for each input length n, Q contains at most n many strings of this length, this table is of polynomial size. Deciding whether a string of length n is in Q can clearly be done in polynomial time, when given such a table. Therefore, $Q \in \text{P}/\text{poly}$.

On the other hand, we claim that Q is not solvable in polynomial time using a constant number of bits as advice. We proceed indirectly, and suppose that there is a polynomial-time algorithm A that decides Q, when given c many bits of advice for each input size, for some fixed constant c. We know that (A, c) appears in the enumeration of all pairs of polynomial-time algorithms and constants. Let $(A_i, c_i) = (A, c)$, for some $i \geq 1$. Then consider the input size n that is used in stage i of the construction of Q. Moreover, let y be the advice string (of length c) that the algorithm A uses to solve inputs of size n. We know that y appeared as some string y_j of length c_i in stage i of the construction of Q. Now consider input x_j. By definition of Q, we know that the algorithm A gives the wrong output for x_j, when using y_j as advice. Thus, A does not solve Q using c many bits of advice. Since A and c were arbitrary, we can conclude that Q is not solvable in polynomial time using a constant number of bits as advice. □

Proposition 32. *There is a parameterized problem $Q \in (\text{FPT}/\text{kernel}) \setminus (\text{FPT}/\text{poly})$.*

Proof. We use a diagonalization argument to construct the problem Q. Let $(A_1, p_1), (A_2, p_2), \ldots$ be an enumeration of all pairs (A_i, p_i) consisting of an fpt-algorithm A_i and a polynomial p_i. We will construct Q in such a way that for each algorithm A_i and each polynomial p_i, there is some input size n, such that for each advice string y of size $p_i(n)$,

the algorithm A_i makes a mistake on some input x of length n when using y as advice. We construct Q in stages: one stage for each (A_i, p_i). In stage i, we make sure that A_i makes such a mistake for the polynomial p_i.

We now describe how to construct Q in a given stage i. Let (A_i, p_i) be the pair consisting of an fpt-algorithm A_i and a polynomial p_i. We may assume without loss of generality that for each such polynomial p_i it holds for all n that $p_i(n) \geq n$. Take an input size n that has not yet been considered in previous stages. Since we have infinitely many input sizes at our disposal, we can always find such an n. Moreover, fix the parameter value $k = p_i(n)$. Then, consider all possible advice strings $y_1, \ldots, y_u$ of length $p_i(n)$, where $u = 2^k$. Also, consider u many (arbitrary, but different) input strings $x_1, \ldots, x_u$ of length n. For each $1 \leq j \leq u$, we let $(x_j, k) \in Q$ if and only if the algorithm A_i, using the advice string y_j, rejects the input string x_j. For all other input strings x of length n, we let $(x, k) \notin Q$. Note that Q contains at most $2^k \leq n$ many pairs (x, k), where x is a string of length n. This completes stage i. For all pairs (n, k) consisting of an input length n and a parameter value k that are not considered in any stage of our construction, we let Q contain no pairs (x, k), where x is a string of length n.

We show that $Q \in$ FPT/kernel. We define the advice for the pair (n, k), consisting of input size n and parameter value k, to be a table consisting of all strings x of length n such that $(x, k) \in Q$. Since for each input length n, Q contains at most 2^k many pairs (x, k), where x is a string of length $n \leq k$, this table is of size $f(k) = O(k2^k)$. Given a pair (x, k) with x of length n, and given such a table for (n, k), deciding whether $(x, k) \in Q$ can clearly be done in fpt-time. Therefore, $Q \in$ FPT/kernel.

On the other hand, we claim that $Q \notin$ FPT/poly. We proceed indirectly, and suppose that there is an fpt-algorithm A that decides Q, when given $p(n)$ many bits of advice for each input size n, for some fixed polynomial p. We know that (A, p) appears in the enumeration of all pairs of fpt-algorithms and polynomials. Let $(A_i, p_i) = (A, p)$, for some $i \geq 1$. Then consider the input size n and the parameter value $k = p(n)$ that are used in stage i of the construction of Q. Moreover, let y be the advice string (of length $p(n)$) that the algorithm A uses to solve inputs of size n. We know that y appeared as some string y_j of length $p(n)$ in stage i of the construction of Q. Now consider input x_j. By definition of Q, we know that the algorithm A gives the wrong output for (x_j, k), when using y_j as advice. Thus, A does not solve Q using $p(n)$ many bits of advice. Since A and p were arbitrary, we can conclude that $Q \notin$ FPT/poly. □

As a consequence of the above results, for the case of the parameterized complexity class FPT the following relations hold.

Corollary 33. *It holds that:*

- FPT $\subsetneq$ FPT/poly $\subsetneq$ FPT/fpt*;*
- FPT $\subsetneq$ FPT/kernel $\subsetneq$ FPT/fpt*;*
- FPT/poly $\not\subseteq$ FPT/kernel*; and*
- FPT/kernel $\not\subseteq$ FPT/poly.

In fact, this picture can be generalized to arbitrary (decidable) classes that contain FPT.

Proposition 34. *For every decidable parameterized complexity class C, there is a parameterized problem $Q \in$ (FPT/poly) $\setminus$ (C/kernel) and a parameterized problem $Q' \in$ (FPT/kernel) $\setminus$ (C/poly).*

Proof. The proofs of Proposition 31 and 32 can straightforwardly be modified to show this result, by using an enumeration of all algorithms (rather than enumerating all polynomial-time or fpt-time algorithms). □

Corollary 35. *For every decidable parameterized complexity class C such that* FPT $\subseteq C$*, it holds that:*

- $C \subsetneq C$/poly $\subsetneq C$/fpt*;*
- $C \subsetneq C$/kernel $\subsetneq C$/fpt*;*
- C/poly $\not\subseteq C$/kernel*; and*
- C/kernel $\not\subseteq C$/poly.

Finally, we show that (under various complexity-theoretic assumptions), the class para-NP is not contained in any of the classes $C \subseteq \mathrm{XP}/\mathrm{xp}$.

Observation 36. *If* para-NP $\subseteq \mathrm{XP}/\mathrm{slice}$*, then* $\mathrm{P} = \mathrm{NP}$.

Proposition 37. *If* para-NP $\subseteq \mathrm{XP}/\mathrm{kernel}$*, then* $\mathrm{P} = \mathrm{NP}$.

Proof. Consider the para-NP-complete language $\mathrm{SAT}_1 = \{ (\varphi, 1) : \varphi \in \mathrm{SAT} \}$. By our assumption that para-NP $\subseteq$ XP/kernel, we get that $\mathrm{SAT}_1 \in \mathrm{XP}/\mathrm{kernel}$. That is, there is some computable function f such that for each $n \in \mathbb{N}$ there exists some $\alpha(n) \in \Sigma^*$ of size $f(k)$ with the property that for each instance $(x, 1)$ of SAT_1 with $|x| = n$, we can decide in fpt-time, given $(x, \alpha(n, 1), 1)$, whether $(x, 1)$ is a yes-instance of SAT_1. In other words, SAT is solvable in polynomial time using a constant number of bits of advice for each input size n.

Then, by self-reducibility of SAT, in polynomial time, using constant-size advice, we can also construct an algorithm A that computes a satisfying assignment of a propositional formula, if it exists, and fails otherwise. We now show how to use this to construct an algorithm B that solves SAT in polynomial time. The idea is the following. Since we only need a constant number of bits of advice for algorithm A, we can simply try out all (constantly many) possible advice strings in a brute force fashion. Given a propositional formula φ, algorithm B iterates over all (constantly many) possible advice strings. For each such string α, algorithm B simulates algorithm A on φ using α. If A outputs a truth assignment for φ, algorithm B verifies whether this assignment satisfies φ. If it does, then clearly φ is satisfiable, and so B accepts φ. If it does not, B continues with the next possible advice string. If for no advice string, the simulation of A outputs a truth assignment, B rejects φ. In this case, we can safely conclude that φ is unsatisfiable. Since at least one advice string leads to correct behavior of A, we know that if φ were satisfiable, a satisfying assignment would have been constructed in some simulation of A. Thus, we can conclude that $\mathrm{P} = \mathrm{NP}$. □

Proposition 38. *If* para-NP $\subseteq \mathrm{XP}/\mathrm{xp}$*, then the PH collapses to the second level.*

Proof. Assume that para-NP $\subseteq \mathrm{XP}/\mathrm{xp}$. Consider the NP-complete language $\mathrm{SAT}_1 = \{ (\varphi, 1) : \varphi \in \mathrm{SAT} \}$. This is then in P/poly. Thus, NP $\subseteq \mathrm{P}/\mathrm{poly}$. By the Karp-Lipton Theorem [19], it follows that $\mathrm{PH} = \Sigma_2^{\mathrm{p}}$. □

Corollary 39. *It holds that* para-NP $\not\subseteq \mathrm{FPT}/\mathrm{fpt}$*,* para-NP $\not\subseteq \mathrm{XP}/\mathrm{fpt}$*,* para-NP $\not\subseteq \mathrm{FPT}/\mathrm{poly}$ *and* para-NP $\not\subseteq \mathrm{XP}/\mathrm{poly}$*, unless the PH collapses to the second level.*

For a more detailed treatment of the Polynomial Hierarchy (PH), we refer to Section 5.

4 Relation to Parameterized Knowledge Compilation

In the previous sections, we have defined various non-uniform complexity classes that come up in the framework of parameterized complexity. In this section, we show how some of these classes are related to the framework of parameterized knowledge compilation.

Knowledge compilation is a technique of dealing with intractable problems where part of the input stays stable for an extended amount of time. However, it turns out that many important problems do not lead to positive compilation results. Parameterized knowledge compilation refers to the use of the parameterized complexity framework to knowledge compilation, with the aim of increasing the possibility of positive compilation results [5].

However, unsurprisingly, there are also problems that cannot be compiled even in the parameterized setting. Most of these negative results are in fact conditional on some non-uniform parameterized complexity classes being different. We give an overview of the relation between parameterized incompilability results and non-uniform parameterized complexity classes. In addition, we define some new non-uniform parameterized complexity classes that can be used to get additional (conditional) parameterized incompilability results.

The (parameterized) compilability framework that we use is based on the work by Cadoli et al. [3] and Chen [5]. Relating non-uniform parameterized complexity to the more recent framework of compilability by Chen [6] remains a topic for future research.

4.1 Parameterized Knowledge Compilation

We begin with reviewing the basic notions from (parameterized) knowledge compilability theory.

4.1.1 Compilability

We provide some basic notions from the theory of compilability. For more details, we refer to the work of Cadoli et al. [3]. In the following, we fix an alphabet Σ.

Definition 40. *A* knowledge representation formalism (KRF) *is a subset of* $\Sigma^* \times \Sigma^*$.

Definition 41. *We say that a function* $f : \Sigma^* \to \Sigma^*$ *is* poly-size *if there exists a constant* c *such that for each* $(x, k) \in \Sigma^*$ *it holds that* $|f(x)| \leq n^c$*, where* $n = |x|$.

Definition 42. *Let* C *be a complexity class. A KRF* F *belongs to* comp-C *if there exist (1) a computable poly-size function* $f : \Sigma^* \to \Sigma^*$ *and a KRF* F' *in* C *such that for all pairs* $(x, y) \in \Sigma^* \times \Sigma^*$ *it holds that* $(x, y) \in F$ *if and only if* $(f(x), y) \in F'$.

Note that, unlike the original definition by Cadoli et al. [3], we require the compilation function f to be computable. This is a reasonable requirement for practically useful compilability results. To the best of our knowledge, there are no natural problems where this distinction (between computable and possibly uncomputable compilation functions) makes a difference.

Definition 43. *Let* C *be a complexity class. A KRF* F *belongs to* nucomp-C *if there exist (1) a poly-size function* $f : \Sigma^* \times 1^* \to \Sigma^*$ *and a KRF* F' *in* C *such that for all pairs* $(x, y) \in \Sigma^* \times \Sigma^*$ *it holds that* $(x, y) \in F$ *if and only if* $(f(x, 1^{|y|}), y) \in F'$.

Definition 44. *A KRF* F *is nucomp-reducible to a KRF* F' *(denoted by* $F \leq_{\text{nucomp}} F'$*) if there exist poly-size functions* $f_1, f_2 : \Sigma^* \times 1^* \times \to \Sigma^*$*, an poly-time function* $g : \Sigma^* \times \Sigma^* \to \Sigma^*$*, such that for all pairs* $(x, y) \in \Sigma^* \times \Sigma^*$ *it holds that* $(x, y) \in F$ *if and only if* $(f_1(x, 1^{|y|}), g(f_2(x, 1^{|y|}), y)) \in F'$.

Note that we do not require the compilation function f witnessing membership in nucomp-C, and the compilation functions f_1 and f_2, nucomp-reducibility, to be computable. The reason for this is that the class nucomp-C and the corresponding nucomp-reductions are intended to show incompilability results, which are even stronger when possibly uncomputable compilation functions are excluded.

We will use the following notation of *compilation problems*. Consider the compilation problem F given as follows.

F
Offline instance: $x \in \Sigma^*$ with property P_1.
Online instance: $y \in \Sigma^*$ with property P_2.
Question: does (x, y) have property P_3?

We use this notation to denote the KRF $F = \{ (x, y) \in \Sigma^* \times \Sigma^* : (x, y)$ satisfies properties $P_1, P_2, P_3 \}$.

Definition 45. *Let* L *be a decision problem. Then the KRF* ϵL *is defined as follows:*

$$\epsilon L = \{\epsilon\} \times L = \{ (\epsilon, x) : x \in L \},$$

where ϵ *is the empty string.*

Proposition 46 ([3, Theorem 2.9])**.** *Let* C *be a complexity class (that is closed under polynomial-time reductions). Let* S *be a problem that is complete for* C *(under polynomial-time reductions). Then* ϵS *is complete for* nucomp-C *under (polynomial-time) nucomp-reductions.*

4.1.2 Parameterized Compilability

Next, we revisit the basic notions of parameterized compilability. For more details, we refer to the work of Chen [5].

Definition 47. *A* parameterized knowledge representation formalism (PKRF) *is a subset of* $\Sigma^* \times \Sigma^* \times \Sigma^*$. *A PKRF can also be seen as a parameterized problem by pairing together the first two strings of each triple.*

Definition 48. *We say that a function $f : \Sigma^* \times \mathbb{N} \to \Sigma^*$ is* fpt-size *if there exists a constant c and a computable function $h : \mathbb{N} \to \mathbb{N}$ such that for each $(x, k) \in \Sigma^* \times \mathbb{N}$ it holds that $|f(x, k)| \leq h(k)n^c$, where $n = |x|$.*

Definition 49. *Let C be a parameterized complexity class. A PKRF F belongs to* par-comp-C *if there exist an fpt-size computable function $f : \Sigma^* \times \mathbb{N} \to \Sigma^*$ and a PKRF F' in C such that for all triples $(x, y, k) \in \Sigma^* \times \Sigma^* \times \mathbb{N}$ and all natural numbers $m \geq |y|$ it holds that $(x, y, k) \in F$ if and only if $(f(x, k), y, k) \in F'$.*

Note that we require the compilation function f witnessing membership in par-comp-C to be computable. This is a reasonable requirement for practically useful compilability results.

Definition 50. *Let C be a parameterized complexity class. A PKRF F belongs to* par-nucomp-C *if there exist an fpt-size function $f : \Sigma^* \times 1^* \times \mathbb{N} \to \Sigma^*$ and a PKRF F' in C such that for all triples $(x, y, k) \in \Sigma^* \times \Sigma^* \times \mathbb{N}$ and all natural numbers $m \geq |y|$ it holds that $(x, y, k) \in F$ if and only if $(f(x, 1^m, k), y, k) \in F'$.*

Observation 51. *Clearly, for each parameterized complexity class C it holds that* par-comp-$C \subseteq$ par-nucomp-C.

Definition 52. *A PKRF F is fpt-nucomp-reducible to a PKRF F' (denoted by $F \leq^{\text{fpt}}_{\text{nucomp}} F'$) if there exist fpt-size functions $f_1, f_2 : \Sigma^* \times 1^* \times \mathbb{N} \to \Sigma^*$, an fpt-time function $g : \Sigma^* \times \Sigma^* \times \mathbb{N} \to \Sigma^*$, and a computable function $h : \mathbb{N} \to \mathbb{N}$ such that for all triples $(x, y, k) \in \Sigma^* \times \Sigma^* \times \mathbb{N}$ and all natural numbers $m \geq |y|$ it holds that $(x, y, k) \in F$ if and only if $(f_1(x, 1^m, k), g(f_2(x, 1^m, k), y, k), h(k)) \in F'$.*

Note that, like the original definition by Chen [5], we do not require the compilation function f witnessing membership in par-nucomp-C, and the compilation functions f_1 and f_2, fpt-nucomp-reducibility, to be computable. The reason for this is that the class par-nucomp-C and the corresponding fpt-nucomp-reductions are intended to show incompilability results, which are even stronger when possibly uncomputable compilation functions are excluded. Moreover, allowing possibly uncomputable compilation functions allows us to establish closer connections to several non-uniform parameterized complexity classes.

We will use the following notation of *parameterized compilation problems*. Consider the parameterized compilation problem F given as follows.

F
Offline instance: $x \in \Sigma^*$ with property P_1.
Online instance: $y \in \Sigma^*$ with property P_2.
Parameter: $f(x, y) = k$.
Question: does (x, y) have property P_3?

We use this notation to denote the PKRF $F = \{ (x, y, k) \in \Sigma^* \times \Sigma^* \times \mathbb{N} : k = f(x, y), (x, y, k)$ satisfies properties $P_1, P_2, P_3 \}$.

Definition 53. *Let L be a parameterized decision problem. Then the PKRF ϵL is defined as follows:*

$$\epsilon L = \{ (\epsilon, x, k) : (x, k) \in L \},$$

where ϵ is the empty string.

The following result is given without proof by Chen [5]. We provide a proof. The main lines of the proof follow that of the proof of Proposition 46 (cf. [3, Theorem 2.9]).

Proposition 54 ([5, Theorem 17])**.** *Let C be a parameterized complexity class. If the parameterized problem Q is complete for C (under fpt-reductions), then ϵQ is complete for* par-nucomp-C *(under fpt-nucomp-reductions).*

Proof. Let Q be a parameterized problem that is C-complete under fpt-reductions. Moreover, let A be an arbitrary PKRF in par-nucomp-C. Then there exist an fpt-size function $f : \Sigma^* \times 1^* \times \mathbb{N} \to \Sigma^*$ and a PKRF Q' in C such that for all triples $(x, y, k) \in \Sigma^* \times \Sigma^* \times \mathbb{N}$ and all natural numbers $m \geq |y|$ it holds that:

$$(x, y, k) \in A \qquad \text{if and only if} \qquad (f(x, 1^m, k), y, k) \in Q'.$$

But $Q' \in C$, so it can be reduced to Q by means of an fpt-reduction R:

$$(x, y, k) \in Q' \qquad \text{if and only if} \qquad R(x, y, k) \in Q.$$

We now define an fpt-nucomp-reduction from A to ϵQ. We specify fpt-size functions f_1, f_2, an fpt-time function g, and a computable function h, as follows:

$$\begin{aligned} f_1(x, 1^m, k) &= \epsilon, \\ f_2(x, 1^m, k) &= f(x, 1^m, k), \\ g(x', y, k) &= \pi_1(R(x', y, k)), \text{ and} \\ h(k) &= \pi_2(R(x', y, k)). \end{aligned}$$

Here π_i represents projection to the i-th element of a tuple. Since R is an fpt-reduction, we know that $\pi_2(R(x', y, k))$ only depends on k, and can thus be expressed as a function h that only depends on k.

We then get that for each $m \geq |y|$:

$$\begin{aligned} (x, y, k) \in A \quad & \text{if and only if} \quad (f(x, 1^m, k), y, k) \in Q' \\ & \text{if and only if} \quad R(f(x, 1^m, k), y, k) \in Q \\ & \text{if and only if} \quad (\epsilon, R(f(x, 1^m, k), y, k)) \in \epsilon Q \\ & \text{if and only if} \quad (f_1(x, 1^m, k), g(f_2(x, 1^m, k), y, k), h(k)) \in \epsilon Q. \end{aligned}$$

Thus, this is a correct fpt-nucomp-reduction from A to ϵQ. Since A was an arbitrary PKRF in par-nucomp-C, we can conclude that ϵQ is par-nucomp-C-complete. □

4.2 (Conditional) incompilability results

To exemplify what role non-uniform parameterized complexity can play in parameterized incompilability results, we give an example of a parameterized problem that does not allow an fpt-size compilation, unless W[1]/fpt $\subseteq$ FPT/fpt.

As main example of a compilation problem in this paper, we will take the following problem CONSTRAINED-CLIQUE. The definition of this problem is analogous to the definition of the Constrained Vertex Cover problem in the seminal paper by Cadoli et al. [3].

CONSTRAINED-CLIQUE
Offline instance: a graph $G = (V, E)$.
Online instance: two subsets $V_1, V_2 \subseteq V$ of vertices, and a positive integer $u \geq 1$.
Question: is there a clique $C \subseteq V$ in G of size u such that $C \cap V_1 = \emptyset$ and $V_2 \subseteq C$?

Below, we will use the following parameterization of this problem to show the relation between parameterized compilability and non-uniform parameterized complexity. The parameter of this problem is the number of vertices that have to be added to V_2 to create a clique of size u.

CONSTRAINED-CLIQUE(sol.-size)
Offline instance: a graph $G = (V, E)$.
Online instance: two subsets $V_1, V_2 \subseteq V$ of vertices, and a positive integer $u \geq 1$.
Parameter: $k = u - |V_2|$.
Question: is there a clique $C \subseteq V$ in G of size u such that $C \cap V_1 = \emptyset$ and $V_2 \subseteq C$?

We begin by showing that CONSTRAINED-CLIQUE(sol.-size) is par-nucomp-W[1]-complete. In order to do so, we will use the W[1]-compete problem MULTI-COLORED CLIQUE (for short: MCC) [11]. Instances of MCC are tuples (V, E, k), where k is a positive integer, V is a finite set of vertices partitioned into k subsets $V_1, \ldots, V_k$, and (V, E) is a simple graph. The parameter is k. The question is whether there exists a k-clique in (V, E) that contains a vertex in each V_i.

Proposition 55. CONSTRAINED-CLIQUE(sol.-size) *is* par-nucomp-W[1]*-complete. Moreover, hardness holds already for the restricted case where $V_2 = \emptyset$ (and so $u = k$).*

Proof. To show membership in par-nucomp-W[1], it suffices to show membership in W[1]. We give an fpt-reduction to CLIQUE, which is contained in W[1]. Let $(G, (V_1, V_2, u), k)$ be an instance of CONSTRAINED-CLIQUE(sol.-size). The reduction outputs the following instance (G', k) of CLIQUE. The graph G' is obtained from G by removing all vertices in V_1, removing all vertices in V_2, and all vertices that are not connected to each vertex $v \in V_2$.

To show hardness for par-nucomp-W[1], we give a fpt-nucomp-reduction from ϵMCC to CONSTRAINED-CLIQUE(sol.-size). Since MCC is W[1]-hard, we know that the problem ϵMCC is par-nucomp-W[1]-hard (under fpt-nucomp-reductions). We specify fpt-size functions f_1, f_2, a computable function h and an fpt-time function g such that for each instance (δ, G, k) of ϵMCC it holds that for each $m \geq \|G\|$:

$$(\delta, G, k) \in \epsilon\text{MCC} \text{ if and only if } (f_1(\delta, 1^m, k), g(f_2(\delta, 1^m, k), G, k), h(k)) \in \text{CONSTRAINED-CLIQUE(sol.-size)}.$$

Let (δ, G, k) be an instance of ϵMCC, where $G = (V, E)$, where the set V of vertices is partitioned into subsets $V_1, \ldots, V_k$, and where $V_i = \{v_1^i, \ldots, v_n^i\}$, for each $1 \leq i \leq k$. We define $f_1(\delta, 1^m, k)$ to be the graph $G' = (V', E')$. Here we let $V' = \bigcup_{1 \leq i \leq k} V^i \cup \bigcup_{1 \leq i < j \leq k} W^{i,j}$, where for each $1 \leq i \leq k$ we let $V^i = \{\, v_\ell^i : 1 \leq \ell \leq m \,\}$, and for each $1 \leq i < j \leq k$ we let $W^{i,j} = \{\, w_{i,\ell_1,j,\ell_2} : 1 \leq \ell_1, \ell_2 \leq m \,\}$. We let E' consist of the following edges. First, we connect all vertices between different sets V^i, i.e., for each $1 \leq i < j \leq k$, we connect each vertex $v_\ell^i \in V^i$ with each vertex $v_{\ell'}^j$. Next, we connect all vertices between different sets $W^{i,j}$, i.e., for each $1 \leq i < j \leq k$ and each $1 \leq i' < j' \leq k$, we connect each vertex w_{i,ℓ_1,j,ℓ_2} with each vertex $w_{i',\ell_1',j',\ell_2'}$. Then, we connect vertices in $W^{i,j}$ and in V^b for $b \neq i$ and $b \neq j$, i.e., for each $1 \leq i < j \leq k$, and each $1 \leq b \leq k$ such that $b \notin \{i, j\}$, we connect each vertex w_{i,ℓ_1,j,ℓ_2} with each vertex v_ℓ^b. Finally, we describe the edges between vertices in $W^{i,j}$ and vertices in $V^i \cup V^j$. Let $1 \leq i < j \leq k$, and let $1 \leq \ell_1, \ell_2 \leq m$. Then we connect w_{i,ℓ_1,j,ℓ_2} with $v_{\ell_1}^i$ and with $v_{\ell_2}^j$. Then, we define $h(k) = k' = k + \binom{k}{2}$. We define $f_2(\delta, 1^m, k) = 1^m$. Finally, we let $g(1^m, G, k) = (U_1, U_2, u)$. Here we let $U_1 = \{\, w_{i,\ell_1,j,\ell_2} : 1 \leq i < j \leq k, (\ell_1 > n \vee \ell_2 > n) \,\} \cup \{\, w_{i,\ell_1,j,\ell_2} : 1 \leq i < j \leq k, \{v_{\ell_1}^i, v_{\ell_2}^j\} \notin E \,\}$. Moreover, we let $U_2 = \emptyset$, and we let $u = |U_2| + k' = k'$. Clearly f_1 and f_2 are fpt-size functions, g is an fpt-time computable function, and h is a computable function.

All that remains to show is that G has a (multi-colored) clique of size k if and only if G' has a clique of size u that contains no vertex in V_1. One direction is easy. Let $C = \{v_{\ell_1}^1, \ldots, v_{\ell_k}^k\}$ be a multicolored clique in G. Then the set $C \cup \{\, w_{i,\ell_i,j,\ell_j} : 1 \leq i < j \leq k \,\}$ is a clique in G' of size k' that contains no vertices in V_1. Conversely, assume that G' has a clique C' of size k' that contains no vertices in V_1. We know that the set V' of vertices of G' consists of k' many subsets for which holds that there are no edges between any two vertices of the same subset. Namely, these subsets are $V^1, \ldots, V^k, W^{1,2}, \ldots, W^{k-1,k}$. Therefore C' must contain one vertex from each of these subsets. Let $C = C' \cap V$. We know that C contains no vertex of the form v_ℓ^i for $n < \ell \leq m$, because for each $n < \ell \leq m$, the vertex v_ℓ^i is not connected to any vertex $W = \bigcup_{1 \leq i < j \leq k} W^{i,j}$ that is not contained in V_1. Therefore, we know that $C \subseteq V$ and $|C| = k$. Let $C = \{v_{\ell_1}^1, \ldots, v_{\ell_k}^k\}$, where for each $1 \leq i \leq k$ we have $C \cap V_i = \{v_{\ell_i}^i\}$. By definition of G', we know that C' must contain for each $1 \leq i < j \leq k$ the vertex w_{i,ℓ_i,j,ℓ_j}. Then, because we know that w_{i,ℓ_i,j,ℓ_j} is not in V_1, by construction of V_1 we know that $\{v_{\ell_i}^i, v_{\ell_j}^j\} \in E$. Therefore, we can conclude that C' is a (multicolored) clique in V. This concludes the correctness proof of our fpt-nucomp-reduction from ϵMCC to CONSTRAINED-CLIQUE(sol.-size). □

We can then use this completeness result to relate fpt-size compilability of this problem to an inclusion between non-uniform parameterized complexity classes. In order to do so, we will first need the following results. These results were already given without proof by Chen [5]. We provide a proof here.

Proposition 56 ([5, Proposition 16])**.** *Let C be a parameterized complexity class. If a parameterized problem S is in C/*fpt*, then $\epsilon S \in$* par-nucomp-C.

Proof. Let S be a parameterized problem in C/fpt. Then, by definition, there exists an fpt-size function f and a parameterized problem $S' \in C$ such that:

$$(y, k) \in S \qquad \text{if and only if} \qquad (f(1^{|y|}, k), y, k) \in S'.$$

We show that $\epsilon S \in$ par-nucomp-C. We specify an fpt-size function f' such that:

$$(\epsilon, y, k) \in \epsilon S \qquad \text{if and only if} \qquad (f'(\epsilon, 1^{|y|}, k), y, k).$$

We let $f'(x, y, k) = f(1^{|y|}, k)$. It is straightforward to modify f' in such a way that it works for each $m \geq |y|$, rather than just for $|y|$. This witnesses that $\epsilon S \in$ par-nucomp-C. □

The proof of the next proposition follows the main lines of the proof of its counterpart in classical compilability (cf. [3, Theorem 2.12]).

Proposition 57 ([5, Theorem 18])**.** *Let C and C' be parameterized complexity classes that are closed under fpt-reductions and that have complete problems. The inclusion* par-nucomp-$C \subseteq$ par-nucomp-C' *holds if and only if the inclusion $C/\text{fpt} \subseteq C'/\text{fpt}$ holds.*

Proof. First, we show that $C/\text{fpt} \subseteq C'/\text{fpt}$ implies that par-nucomp-$C \subseteq$ par-nucomp-C'. Suppose that $C/\text{fpt} \subseteq C'/\text{fpt}$. Moreover, let S be a problem that is C-complete under fpt-reductions. Then, by Proposition 54, ϵS is par-nucomp-C-complete. Also, since $S \in C/\text{fpt}$ and $C/\text{fpt} \subseteq C'/\text{fpt}$, we know that $S \in C'/\text{fpt}$. Therefore, by Proposition 56, $\epsilon S \in$ par-nucomp-C'. Then, since there is a par-nucomp-C-complete problem that is in par-nucomp-C', we get the inclusion par-nucomp-$C \subseteq$ par-nucomp-C'.

Conversely, we show that par-nucomp-$C \subseteq$ par-nucomp-C' implies that $C/\text{fpt} \subseteq C'/\text{fpt}$. Suppose that par-nucomp-$C \subseteq$ par-nucomp-C'. Let S be an arbitrary problem in C/fpt. We show that $S \in C'/\text{fpt}$. By definition, there exists an fpt-size function f and a PRKF $S' \in C$ such that:

$$(y, k) \in S \qquad \text{if and only if} \qquad (f(1^{|y|}, k), k) \in S'.$$

From this it follows that the PKRF ϵS is in par-nucomp-C. Therefore, it is also in par-nucomp-C'. This means that there is an fpt-size function f' and a PKRF $S'' \in C'$ such that for all instances (x, y, k):

$$(x, y, k) \in \epsilon S \qquad \text{if and only if} \qquad (f'(x, 1^{|y|}, k), y, k) \in S''.$$

Then, we get that:

$$(y, k) \in S \qquad \text{if and only if} \qquad (\epsilon, y, k) \in \epsilon S \qquad \text{if and only if} \qquad (f'(\epsilon, 1^{|y|}, k), y, k) \in S''.$$

Since $f'(\epsilon, 1^{|y|}, k)$ is an fpt-size function that depends on $1^{|y|}$ and k alone, and since $S'' \in C'$, we conclude that $S \in C'/\text{fpt}$. Since S was an arbitrary problem in C/fpt, we can conclude that $C/\text{fpt} \subseteq C'/\text{fpt}$. □

Corollary 58. par-nucomp-W[1] $\subseteq$ par-nucomp-FPT *if and only if* W[1]/fpt $\subseteq$ FPT/fpt.

From this, we get the following result about CONSTRAINED-CLIQUE(sol.-size).

Proposition 59. CONSTRAINED-CLIQUE(sol.-size) *is not in* par-nucomp-FPT*, and so it is not in* par-comp-FPT*, unless* W[1]/fpt $\subseteq$ FPT/fpt.

4.3 Restricting the instance space

In order to connect the parameterized compilability of another natural parameterized variant of CONSTRAINED-CLIQUE (that we will define in Section 4.4) to non-uniform parameterized complexity, we will need some additional non-uniform parameterized complexity classes. In this section, we develop these classes.

Concretely, we will define two new parameterized complexity classes: few-NP and nu-few-NP. In order to define these classes, we will consider a particular type of functions, and a parameterized decision problem based on such functions.

Note that these classes are not directly connected to the class FEWP consisting of all NP problems with a polynomially-bounded number of solutions [1], as we will see in more detail below. The classes few-NP and nu-few-NP consist of problems that have few NP instances (each of which can have many solutions).

Definition 60 (generators)**.** *We say that a function $\gamma : \mathbb{N}^3 \to \Sigma^*$ is a* (SAT instance) generator *if for each $(n, \ell, k) \in \mathbb{N}^3$ it holds that:*

- *if $1 \leq \ell \leq n^k$, then $\gamma(n, \ell, k)$ is a SAT instance, i.e., a propositional formula;*

- *otherwise $\gamma(n, \ell, k)$ is the trivial SAT instance $\emptyset$.*

We say that a generator γ is nice *if for each (n, ℓ, k) the formula $\gamma(n, \ell, k)$ has exactly n variables. We say that a generator is a* 3CNF generator *if all the (nontrivial) instances that it generates are propositional formulas in 3*CNF*.*

Moreover, we say that a generator is (uniformly) fpt-time computable *if there exists an algorithm A, a computable function f and a constant c such that for each $(n, \ell, k) \in \mathbb{N}^3$ the algorithm A computes $\gamma(n, \ell, k)$ in time $f(k)n^c$.*

We say that a generator is non-uniformly fpt-time computable *if there exist a computable function f and a constant c such that for each $(n, k) \in \mathbb{N}^2$, there exists an algorithm $A_{(n,k)}$ that for each $\ell \in \mathbb{N}$ computes $\gamma(n, \ell, k)$ in time $f(k)n^c$ (this corresponds to the non-uniformity notion of fpt-size advice).*

Definition 61. *Let γ be a generator. We define the parameterized decision problem* FEWSAT$_\gamma$ *as follows.*

FEWSAT$_\gamma$
Instance: $(n, \ell, k) \in \mathbb{N}^3$, *where n is given in unary and ℓ, k are given in binary.*
Parameter: k.
Question: *is $\gamma(n, \ell, k)$ satisfiable?*

Definition 62. *Let γ be a generator. We define the parameterized decision problem* FEWUNSAT$_\gamma$ *as follows.*

FEWUNSAT$_\gamma$
Instance: $(n, \ell, k) \in \mathbb{N}^3$, *where n is given in unary and ℓ, k are given in binary.*
Parameter: k.
Question: *is $\gamma(n, \ell, k)$ unsatisfiable?*

Definition 63. *We define the following parameterized complexity class* few-NP*:*

$$\text{few-NP} = [\{ \text{FEWSAT}_\gamma : \gamma \text{ is a uniformly fpt-time computable nice 3CNF generator } \}]_{\text{fpt}}.$$

By using Definition 1, we could obtain the non-uniform variant few-NP/fpt of this parameterized complexity class few-NP. However, we believe that the following definition of nu-few-NP captures a more natural non-uniform version of the class few-NP.

Definition 64. *We define the following parameterized complexity class* nu-few-NP*:*

$$\text{nu-few-NP} = [\{ \text{FEWSAT}_\gamma : \gamma \text{ is a non-uniformly fpt-time computable nice 3CNF generator } \}]_{\text{fpt}}.$$

The intuition behind the classes few-NP and nu-few-NP is the following. Both the classes W[t] of the Weft-hierarchy and the classes few-NP and nu-few-NP contain problems that are restrictions of problems in NP. For the classes W[t] the set of possible witnesses is restricted from $2^{O(n)}$ many to $n^{O(k)}$ many. The classes few-NP and nu-few-NP are based on a dual restriction. For these classes, not the set of possible witnesses is restricted, but the set of instances (for each n, k) is restricted from $2^{O(n)}$ many to $n^{O(k)}$ many. In principle, such a restriction could be exploited by fpt-algorithms to solve the restricted variants of the NP problems.

We illustrate the notion of (fpt-time computable) 3CNF generators using the following example.

Example 65. For each $n \in \mathbb{N}$, there are 2^{8n^3} possible different 3CNF formulas over the variables $x_1, \ldots, x_n$, each of which can thus be described by $8n^3$ bits. On the other hand, for each $(n, k) \in \mathbb{N}^2$, each integer $1 \leq \ell \leq n^k$ can be described by $k \log n$ bits. We consider the following (abstract) example of an fpt-time computable 3CNF generator. Using an *pseudorandom generator*, we construct a function that for each $(n, k) \in \mathbb{N}^2$ and each bitstring of length $k \log n$ (representing the number ℓ) produces a seemingly random bitstring of length $8n^3$ that is interpreted as a 3CNF formula over the variables $x_1, \ldots, x_n$. ⊣

Before we put these newly introduced non-uniform parameterized complexity classes to use by employing them to characterize the (in)compilability of certain parameterized compilation problems, we make a few digressions and provide some alternative characterizations of few-NP and nu-few-NP and relate them to other non-uniform parameterized complexity classes.

4.3.1 Normalization results

It will be useful to develop some normalization results for the classes few-NP and nu-few-NP. In this section, we will do so.

Proposition 66. *Let γ be a uniformly fpt-time computable 3*CNF *generator that is not nice. Then* $\text{FEWSAT}_\gamma \in$ few-NP.

Proof. We construct a nice 3CNF generator γ' and we give an fpt-reduction from FEWSAT_γ to $\text{FEWSAT}_{\gamma'}$. Let f be a computable function and let c be a constant such that for each (n, ℓ, k), $\gamma(n, \ell, k)$ can be computed in time $f(k)n^c$. We consider the function π that is defined by $\pi(n', k) = \lfloor \sqrt[c]{n'/f(k)} \rfloor$, i.e., for each $(n', k) \in \mathbb{N}^2$ it holds that $n' = f(k)n^c$ where $n = \pi(n', k)$. This function π is fpt-time computable. Also, for each $(n', k) \in \mathbb{N}^2$ it holds that $\pi(n', k) \leq n'$.

Next, we construct the uniformly fpt-time computable 3CNF generator γ' as follows. We describe the algorithm that computes γ'. On input (n', ℓ, k), the algorithm first computes $n = \pi(n', k)$. Then, it computes $\varphi = \gamma(n, k, \ell)$. The formula φ is of size at most $f(k)n^c$, and thus contains at most $f(k)n^c$ many variables. The algorithm then transform φ into an equivalent formula φ' that has exactly $f(k)n^c$ many variables by adding dummy variables. Finally, the algorithm returns φ'.

All that remains is to specify an fpt-reduction from FEWSAT_γ to $\text{FEWSAT}_{\gamma'}$. Given an instance (n, ℓ, k) of FEWSAT_γ, the reduction returns the instance $(f(k)n^c, \ell, k)$ of $\text{FEWSAT}_{\gamma'}$. Clearly, this is computable in fpt-time. Since we know that $\gamma'(f(k)n^c, \ell, k) \equiv \gamma(n, \ell, k)$, this reduction is correct. □

Proposition 67. *Let γ be a non-uniformly fpt-time computable 3*CNF *generator that is not nice. Then* $\text{FEWSAT}_\gamma \in$ nu-few-NP.

Proof. Completely analogous to the proof of Proposition 66 □

Proposition 68. *Let γ be a uniformly fpt-time computable generator (that is not necessarily a 3*CNF *generator). Then* $\text{FEWSAT}_\gamma \in$ few-NP.

Proof. We construct a uniformly fpt-time computable 3CNF generator γ' as follows. We describe the algorithm that computes γ'. On input (n, ℓ, k), the algorithm first computes $\varphi = \gamma(n, k, \ell)$. Using the well-known Tseitin transformation, the algorithm constructs a formula φ' in 3CNF that is equivalent to φ. The algorithm then returns $\gamma'(n, \ell, k) = \varphi'$.

The generator γ' is a uniformly fpt-time computable 3CNF generator that is not necessarily nice. The identity mapping is then an fpt-reduction from FEWSAT_γ to $\text{FEWSAT}_{\gamma'}$. We know by Proposition 66 that $\text{FEWSAT}_{\gamma'} \in$ few-NP. From this we can conclude that $\text{FEWSAT}_\gamma \in$ few-NP. □

Proposition 69. *Let γ be a non-uniformly fpt-time computable generator (that is not necessarily a 3*CNF *generator). Then* $\text{FEWSAT}_\gamma \in$ nu-few-NP.

Proof. Completely analogous to the proof of Proposition 68 □

4.3.2 Relating few-NP and nu-few-NP to other classes

We situate the classes few-NP and nu-few-NP in the landscape of (non-uniform) parameterized complexity classes that we considered in Section 2. We do so by giving inclusion as well as (conditional and unconditional) non-inclusion results.

Observation 70. *If* P = NP*, then* nu-few-NP $\subseteq$ FPT/fpt.

Proposition 71. *We have the following inclusions:*

1. FPT $\subseteq$ few-NP*;*
2. few-NP $\subseteq$ nu-few-NP*;*
3. few-NP $\subseteq$ para-NP*;*

4. nu-few-NP $\subseteq$ para-NP/fpt;

5. few-NP $\subseteq$ XP/xp;

6. nu-few-NP $\subseteq$ XP/xp;

Proof. Inclusions 1 and 2 are trivial. Inclusion 3 can be shown as follows. For each uniformly fpt-time computable generator γ, the problem FEWSAT_γ can be decided in nondeterministic fpt-time as follows. Firstly, compute $\varphi = \gamma(n, \ell, k)$ in deterministic fpt-time. Then, using nondeterminism, decide if φ is satisfiable.

Next, Inclusion 4 can be shown as follows. For each non-uniformly fpt-time computable generator γ, the problem FEWSAT_γ can be decided in nondeterministic fpt-time using fpt-size advice as follows. Firstly, compute $\varphi = \gamma(n, \ell, k)$ in deterministic fpt-time using fpt-size advice. Then, using nondeterminism, decide if φ is satisfiable.

Inclusion 6 can be shown as follows. For each non-uniformly fpt-time computable generator γ, the problem FEWSAT_γ can be decided in xp-time using xp-size advice as follows. For each (n, k), the advice $\alpha(n, k)$ is a lookup-table T of size $O(n^k)$ that contains for each $1 \leq \ell \leq n^k$ a bit $T_\ell \in \{0, 1\}$ that represents whether $\gamma(n, \ell, k)$ is satisfiable or not.

Then, Inclusion 5 follows directly from Inclusions 2 and 6. □

Observation 72. *By Proposition 38, we then have that* few-NP $\subsetneq$ para-NP*, unless the PH collapses to the second level.*

In fact, we can get an even stronger result.

Proposition 73. *If* para-NP = few-NP*, then* P = NP.

Proof. Consider the NP-complete language $\text{SAT}_1 = \{ (\varphi, 1) : \varphi \in \text{SAT} \}$. We know that SAT_1 is para-NP-complete. By our assumption that para-NP = few-NP, we get that there exists some uniformly fpt-time computable generator γ such that SAT_1 is fpt-reducible to FEWSAT_γ. Call the fpt-reduction that witnesses this R. We then know that there exists a computable function h such that for each instance $(\varphi, 1)$ of SAT_1, the resulting instance $R(\varphi, 1)$ of FEWSAT_γ is of the form (n, ℓ, k'), for some $1 \leq k' \leq h(1)$. Let Q be the restriction of FEWSAT_γ to instances (n, ℓ, k') where $1 \leq k' \leq h(1)$. Because $h(1)$ is a constant, we then have that Q, when considered as a classical (nonparameterized) decision problem, is sparse, i.e., there is a constant c such that for each input size n there are at most n^c many yes-instances of Q of length n. The fpt-reduction R then also witnesses that SAT_1, when considered as a classical (nonparameterized) decision problem, is polynomial-time reducible to Q. Since SAT_1 is NP-complete and Q is sparse, by Mahaney's Theorem [21], it then follows that P = NP. □

Corollary 74. few-NP = para-NP *if and only if* P = NP.

Interestingly, this means that if P $\neq$ NP, then the class few-NP lies strictly between FPT and para-NP.

Proposition 75. *There exists a parameterized problem* $Q \in$ (nu-few-NP) $\setminus$ (XP_{nu}).

Proof (sketch). One can straightforwardly show that the problem constructed in the proof of Proposition 23 is also contained in nu-few-NP. Since this problem was shown not to be in XP_{nu}, the result then follows. □

Remark 76. One could also analogously define a parameterized complexity class slice-few-NP. This class would be based on the problems FEWSAT_γ for those (3CNF) generators γ that are uniformly fpt-time computable for each fixed k (where the fpt-running time is upper bounded by a fixed expression $f(k)n^c$ for all k).

4.3.3 Complete problems

An interesting question is whether the class few-NP has any problems that are complete for it under fpt-reductions. We briefly state an observation that could potentially be used to get more insight into this question.

Observation 77. *If there is some parameterized problem Q that is complete for* few-NP *under fpt-reductions, then there exists some uniformly fpt-time computable nice 3*CNF *generator γ such that* FEWSAT_γ *is* few-NP*-complete (under fpt-reductions).*

4.3.4 Differences between non-uniform variants of few-NP

Above, we (implicitly) provided two different possible definitions of a non-uniformly defined variant of few-NP: few-NP/fpt and nu-few-FPT. We give a few remarks about how they relate to each other.

The difference between few-NP/fpt and nu-few-NP can be described as follows. The class few-NP/fpt consists of all parameterized problems that are **non-uniformly** fpt-reducible to the problem FEWSAT_γ, for some **uniformly** fpt-time computable generator γ. The class nu-few-NP consists of all parameterized problems that are **uniformly** fpt-reducible to the problem FEWSAT_γ, for some **non-uniformly** fpt-time computable generator γ. For the usual parameterized complexity classes (such as FPT), adding non-uniformity (in the form of fpt-size advice) either (i) to reductions or (ii) to the algorithms witnessing membership in a complexity class, yields the same outcome. This is because one can simply "save" the advice in the problem input. However, since for any problem in few-NP, the problem input can only contain $O(k \log n)$ bits, for any fixed (n, k), you cannot "save" the (fpt-size) advice string in the problem input.

As shown above, the class nu-few-NP is the non-uniform variant of few-NP that results from the "circuit non-uniformity" point of view. The class few-NP/fpt is the non-uniform variant of few-NP that results from the "advice non-uniformity" point of view. Interestingly, for the parameterized complexity class few-NP these two different notions of non-uniformity seem to differ. For all other parameterized complexity classes that we have considered, these two notions of non-uniformity give rise to the same (non-uniform) class.

4.3.5 Characterizing few-NP in terms of 3-colorability

To illustrate that for the classes few-NP and nu-few-NP it does not matter what NP-complete "base problem" one chooses, we give a different characterization of these classes in terms of a problem based on 3-colorability of graphs.

Definition 78 (graph generators)**.** *We say that a function $\gamma : \mathbb{N}^3 \to \Sigma^*$ is a* graph generator *if for each $(n, \ell, k) \in \mathbb{N}^3$ it holds that:*

- *if $1 \leq \ell \leq n^k$, then $\gamma(n, \ell, k)$ is a graph;*
- *otherwise $\gamma(n, \ell, k)$ is the trivial empty graph $(\emptyset, \emptyset)$.*

We say that a graph generator γ is nice *if for each (n, ℓ, k) the graph $\gamma(n, \ell, k)$ has exactly n vertices.*

Moreover, we say that a generator is (uniformly) fpt-time computable *if there exists an algorithm A, a computable function f and a constant c such that for each $(n, \ell, k) \in \mathbb{N}^3$ the algorithm A computes $\gamma(n, \ell, k)$ in time $f(k)n^c$.*

We say that a generator is non-uniformly fpt-time computable *if there exist a computable function f and a constant c such that for each $(n, k) \in \mathbb{N}^2$, there exists an algorithm $A_{(n,k)}$ that for each $\ell \in \mathbb{N}$ computes $\gamma(n, \ell, k)$ in time $f(k)n^c$.*

Definition 79. *Let γ be a graph generator. We define the parameterized decision problem FEW3COL_γ as follows.*

FEW3COL_γ
Instance: $(n, \ell, k) \in \mathbb{N}^3$*, where n is given in unary and ℓ, k are given in binary.*
Parameter: k.
Question: *is $\gamma(n, \ell, k)$ 3-colorable?*

Using these definitions, we derive the following characterization of few-NP.

Proposition 80. *It holds that* few-NP $= [\{ \text{FEW3COL}_\gamma : \gamma \text{ is a uniformly fpt-time computable graph generator } \}]_{\text{fpt}}$.

Proof. Firstly, we show that for each uniformly fpt-time computable graph generator γ it holds that $\text{FEW3COL}_\gamma \in$ few-NP. Let γ be such a graph generator. We construct the following uniformly fpt-time computable SAT instance generator γ'. We describe the algorithm that computes γ'. On input (n, ℓ, k), the algorithm firstly computes the graph $\gamma(n, \ell, k)$. By using the standard polynomial-time reduction from 3-colorability to propositional satisfiability, we can construct (in polynomial time) a propositional formula φ that is satisfiable if and only if $\gamma(n, \ell, k)$ is 3-colorable. The algorithm then returns $\gamma'(n, \ell, k) = \varphi$. It is then straightforward to verify that the identity mapping is an fpt-reduction from FEW3COL_γ to $\text{FEWSAT}_{\gamma'}$. Therefore, we can conclude that $\text{FEW3COL}_\gamma \in$ few-NP.

Next, we show that for each problem $Q \in$ few-NP, it holds that Q is fpt-reducible to FEW3COL_γ for some uniformly fpt-time computable graph generator γ. For this, it suffices to show that for each uniformly fpt-time computable nice 3CNF generator γ there exists some uniformly fpt-time computable graph generator γ' such that FEWSAT_γ is fpt-reducible to $\text{FEW3COL}_{\gamma'}$.

Let γ be a uniformly fpt-time computable nice 3CNF generator. We construct the following uniformly fpt-time computable graph generator γ'. We describe the algorithm that computes γ'. On input (n, ℓ, k), the algorithm firstly computes the 3CNF formula $\varphi = \gamma(n, \ell, k)$. By using the standard polynomial-time reduction from satisfiability of 3CNF formulas to 3-colorability, we can construct (in polynomial time) a graph G that is 3-colorable if and only if φ is satisfiable. The algorithm then returns $\gamma'(n, \ell, k) = G$. It is then straightforward to verify that the identity mapping is an fpt-reduction from FEWSAT_γ to $\text{FEW3COL}_{\gamma'}$. This concludes our proof. □

Proposition 81. nu-few-NP $= [\{ \text{FEW3COL}_\gamma : \gamma \textit{ is a non-uniformly fpt-time computable graph generator } \}]_{\text{fpt}}$.

Proof. The proof is analogous to the proof of Proposition 80. □

4.3.6 Characterizing few-NP by "filtering" para-NP problems

To illustrate the robustness of the classes few-NP and nu-few-NP even more, we give a different characterization of these classes, in terms of applying "well-behaved efficiently computable filters" to para-NP problems.

We begin with the definition of xp-numberings. These functions will be the foundation of our notion of "filters."

Definition 82. *Let Q be a parameterized problem. We say that an* xp-numbering of Q *is a function $\rho : \Sigma^* \times \Sigma^* \to \mathbb{N}$ for which there exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that:*

- *for each $(n, k) \in \mathbb{N} \times \Sigma^*$, ρ maps each instance (x, k) of Q with $|x| = n$ to a number $0 \leq \ell \leq n^{f(k)}$;*
- *for any instance (x, k) of Q it holds that if $\rho(x, k) = 0$, then $(x, k) \notin Q$; and*
- *for each $(n, k) \in \mathbb{N} \times \Sigma^*$ and any two instances $(x, k), (x', k) \in \Sigma^* \times \Sigma^*$ with $|x| = |x'| = n$ it holds that $\rho(x, k) = \rho(x', k)$ implies that $(x, k) \in Q$ if and only if $(x', k) \in Q$.*

Moreover, we say that an xp-numbering is an fpt-time xp-numbering *if it is computable in fixed-parameter tractable time.*

In the remainder of this section, we will consider the following class of parameterized problems:

$$\{ A \cap B : A \text{ has an fpt-time xp-numbering}, B \in \text{para-NP} \}.$$

Intuitively, in this definition, one can consider fpt-time xp-numberings as efficiently computable well-behaved filters: They are efficiently computable, because they run in fpt-time. They are filters, because they reduce the para-NP set to a set where for each (n, k), all instances can be decided by solving the problem for only $n^{f(k)}$ instances. Finally, they are well-behaved, because one can identify for each instance if it boils down to one of these $n^{f(k)}$ "crucial" instances, and if so, to which one of these instances (indicated by a number $1 \leq \ell \leq n^{f(k)}$).

In fact, we will show that this class coincides with few-NP.

Proposition 83. $\{ A \cap B : A \textit{ has an fpt-time xp-numbering}, B \in \text{para-NP} \} \subseteq$ few-NP.

Proof. Let $Q = A \cap B$ for some problem A that has an fpt-time xp-numbering ρ and some $B \in$ para-NP. We show that $Q \in$ few-NP by showing that Q is fpt-reducible to the problem FEWSAT_γ for some uniformly fpt-time computable generator γ. The main idea of this proof is to encode the concatenation of the inverse computation of the xp-numbering ρ and the non-deterministic check of membership in B into the SAT instance generated for some number $1 \leq \ell \leq n^k$.

Let f be the computable function such that for each input (x, k) with $|x| = n$ it holds that $\rho(x, k) \leq n^{f(k)}$. We will construct the uniformly fpt-time generator γ below. Firstly, we will specify the fpt-reduction R from Q to FEWSAT_γ. Let (x, k) be an instance of Q, with $|x| = n$. We let $R(x, k) = (n, \ell, f(k))$, where $\ell = \rho(x, k) \leq n^{f(k)}$. Next, we will specify the generator γ. Since it is more informative to describe the non-deterministic computation encoded

in the SAT instance that γ outputs, we do so without spelling out the resulting SAT instance. On input (n, ℓ, k), the computation $\gamma(n, \ell, k)$ firstly computes $k' = f^{-1}(k)$, i.e., the k' such that $f(k') = k$. This can be done in deterministic fpt-time. Next, the computation $\gamma(n, \ell, k)$ non-deterministically computes an instance x of length n for which it holds that $\rho(x, k) = \ell$. Since for all instances x, x' with $\rho(x, k) = \rho(x', k)$ it holds that $(x, k) \in A$ if and only if $(x', k) \in A$, we can allow the computation to find any such instance x. Finally, the computation $\gamma(n, \ell, k)$ non-deterministically verifies that $(x, k) \in B$. Since $B \in$ para-NP, this can be done in non-deterministic fpt-time. As explained above, technically, γ encodes this non-deterministic fpt-time computation $\gamma(n, \ell, k)$ in a SAT instance of fpt-size. We get for each instance (x, k) with $|x| = n$ that $(x, k) \in Q$ if and only if $R(x, k) = (n, \ell, f(k)) \in \text{FEWSAT}_\gamma$. Therefore, $Q \in$ few-NP. □

Lemma 84. *Let $Q = A \cap B$ be a parameterized problem, where A is a problem that has an fpt-time xp-numbering, and $B \in$ para-NP. Moreover, let Q' be a parameterized problem that is fpt-reducible to Q. Then $Q' = A' \cap B'$ for some problem A' that has an fpt-time xp-numbering and some $B' \in$ para-NP.*

Proof. We construct A' and B'. Let R be the fpt-reduction from Q' to Q. We let A' be the set consisting of the following instances. We let $(x, k) \in A'$ if and only if $R(x, k) \in A$. We show that A' has an fpt-time xp-numbering, by constructing such an xp-numbering ρ'. We know that A has an fpt-time xp-numbering ρ. On input (x, k), the xp-numbering ρ' returns the value $\ell = \rho(R(x, k))$. Since R is an fpt-reduction, is straightforward to construct a computable function f' such that $\ell \leq n^{f'(k)}$ for each instance (x, k) with $|x| = n$. Moreover, since both R and ρ are fpt-time computable, the xp-numbering ρ' is also fpt-time computable.

Next, we construct the set B', in a similar way. For each instance (x, k), we let $(x, k) \in B'$ if and only if $R(x, k) \in B$. Since $B \in$ para-NP and since para-NP is closed under fpt-reductions, we get that $B' \in$ para-NP.

These definitions then have the consequence that for each instance (x, k) it holds that $(x, k) \in Q'$ if and only if $(x, k) \in A' \cap B'$. Namely, we get that $(x, k) \in Q'$ if and only if $R(x, k) \in Q$, if and only if both $R(x, k) \in A$ and $R(x, k) \in B$, if and only if both $(x, k) \in A'$ and $(x, k) \in B'$, if and only if $(x, k) \in A' \cap B'$. □

Proposition 85. few-NP $\subseteq \{\, A \cap B : A$ *has an fpt-time xp-numbering,* $B \in$ para-NP $\,\}$.

Proof. Let $Q \in$ few-NP. We know that there is an fpt-reduction R from Q to FEWSAT_γ, for some uniformly fpt-time computable generator γ. We show that Q is fpt-reducible to a problem $Q' = A \cap B$, for some problem A that has an fpt-time xp-numbering and some $B \in$ para-NP. By Lemma 84, this suffices to show that Q itself is of the required form.

We specify the fpt-reduction R', the problem A with its fpt-time xp-numbering ρ, and the problem $B \in$ para-NP. We begin with the fpt-reduction R'. On input (x, k) with $|x| = n$, the reduction first computes $R(x, k) = (n', \ell', k')$. Here we know that $k' \leq f(k)$ for some computable function f. Next, it computes $\gamma(n', \ell', k') = \varphi$. The reduction then outputs $R(x, k) = (x, \varphi, \ell', k)$. We let the set A consist of all instances (x, φ, ℓ', k) such that $\gamma(n', \ell', k') = \varphi$, where $R(x, k) = (n', \ell', k')$, i.e., A checks whether φ and ℓ' are computed correctly from (x, k), according to R and γ. Next, we specify the xp-numbering ρ of A. On input (x, φ, ℓ', k), the xp-numbering ρ checks whether $(x, \varphi, \ell', k) \in A$; if so, it outputs $\rho(x, \varphi, \ell', k') = \ell'$; otherwise, it outputs $\rho(x, \varphi, \ell', k') = 0$. Since $\ell' \leq |x|^{f(k)}$, we know that ρ is in fact an xp-numbering. Moreover, ρ is fpt-time computable. Finally, we specify the set $B \in$ para-NP. For each input (x, φ, ℓ', k), we let $(x, \varphi, \ell', k) \in B$ if and only if φ is satisfiable. Clearly, then, $B \in$ para-NP. Now, since A checks whether φ is computed correctly from (x, k) according to R and γ, and since B checks whether φ is satisfiable, we get that $(x, k) \in Q$ if and only if $R'(x, k) \in A \cap B$. This concludes our proof. □

Theorem 86. *The class* few-NP *can be characterized in the following way:*

$$\text{few-NP} = \{\, A \cap B : A \text{ has an fpt-time xp-numbering}, B \in \text{para-NP} \,\}.$$

Proof. The result follows directly from Propositions 83 and 85. □

In an entirely analogous way, we can now derive the following characterization of nu-few-NP, using xp-numberings ρ that are in (the search variant of) FPT/fpt.

Theorem 87. *The class* nu-few-NP *can be characterized in the following way:*

$$\text{nu-few-NP} = \{\, A \cap B : A \text{ has an xp-numbering } \rho \text{ computable by a FPT/fpt algorithm}, B \in \text{para-NP} \,\}.$$

Proof. The proof is entirely analogous to the proofs of Theorem 86, Lemma 84, and Propositions 83 and 85. □

We illustrate this characterization of few-NP in terms of filtering by using it to show that the following example problem is in few-NP. We consider the binary expansion of irrational numbers. For instance, take $\sqrt{2}$. We can compute the first n bits of the binary expansion of $\sqrt{2}$ in time polynomial in n. Let $\text{bits}(n, \sqrt{2})$ denote the first n bits in the binary expansion of $\sqrt{2}$.

Example 88. The following parameterized problem $3\text{SAT}\lfloor\text{dist-}\sqrt{2}\rfloor$ is in few-NP:

$$\begin{aligned} 3\text{SAT}\lfloor\text{dist-}\sqrt{2}\rfloor = \{ \quad & (x,k) \in \{0,1\}^* \times \mathbb{N} : \\ & x \text{ encodes a satisfiable propositional formula}, |x| = n, \\ & \text{the Hamming distance between } x \text{ and } \text{bits}(n,\sqrt{2}) \text{ is at most } k \quad \}. \end{aligned}$$

To see this, we express $3\text{SAT}\lfloor\text{dist-}\sqrt{2}\rfloor$ as a problem of the form $A \cap B$, where A has an fpt-time computable xp-numbering and where $B \in$ para-NP. We let $B = 3\text{SAT} \times \mathbb{N}$, which is clearly in para-NP. Then we describe A and its fpt-time xp-numbering ρ. On input (x,k), with $|x| = n$, we first compute $y = \text{bits}(n,\sqrt{2})$. Then we compute $z = x \oplus y$ (where $\oplus$ denotes the bitwise exclusive-or operator). This can be done in polynomial time. Then, if z has more than k ones, we let $(x,k) \notin A$, and $\rho(x,k) = 0$. Otherwise, if z has at most k ones, we let $(x,k) \in A$ and we compute $\rho(x,k)$ as follows. Let $i_1, \ldots, i_u$, with $u \leq k$, be the indices of the bits in $z = z_1 \ldots z_n$ with ones, i.e., $z_{i_j} = 1$ for all $1 \leq j \leq u$. We can extend this sequence of indices to a sequence $i'_1, \ldots, i'_k$ by adding $k - u$ zeroes to the beginning. We then concatenate $i'_1, \ldots, i'_k$ (each described using $\log n$ bits) into a single bitstring i, and interpret this as a number ℓ. We then let $\rho(x,k) = \ell + 1$. Since each i'_j is described by $\log n$ bits, we have that i is of length $k \log n$, and thus $1 \leq \ell \leq n^k$. Thus ρ is an xp-numbering. ⊣

In fact, there is nothing special about the number $\sqrt{2}$ (at least for this example), except for the fact that the first n bits of its binary expansion are computable in polynomial time. Consequently, for any other real number r for which we can compute the first n bits of its binary expansion in polynomial time (in n), the similarly defined problem $\text{SAT}\lfloor\text{dist-}r\rfloor$ is in few-NP.

4.4 The parameterized compilability of finding small cliques

Using the non-uniform parameterized complexity classes defined in the previous section, we can relate the parameterized compilability of several problems to (non-)inclusion properties in the landscape of non-uniform parameterized complexity. We use another natural parameterized variant of CONSTRAINED-CLIQUE as an example to illustrate this.

We consider the following parameterization of CONSTRAINED-CLIQUE.

CONSTRAINED-CLIQUE(constr.-size)
Offline instance: a graph $G = (V, E)$.
Online instance: two subsets $V_1, V_2 \subseteq V$ of vertices, and a positive integer $u \geq 1$.
Parameter: $k = |V_1| + |V_2|$.
Question: is there a clique $C \subseteq V$ in G of size u such that $C \cap V_1 = \emptyset$ and $V_2 \subseteq C$?

We now characterize the parameterized compilation complexity of CONSTRAINED-CLIQUE(constr.-size) using the parameterized complexity classes nu-few-FPT and FPT/fpt. In order to do so, we consider the following auxiliary parameterized compilation problem.

SATQUERY(query-size)
Offline instance: a propositional formula φ.
Online instance: a set L of literals.
Parameter: $k = |L|$.
Question: is $\varphi \wedge \bigwedge_{l \in L} l$ satisfiable?

Proposition 89. *The problem* SATQUERY(query-size) *is equivalent to the restriction of* SATQUERY(query-size) *to offline instances in 3*CNF *(under fpt-nucomp-reductions).*

Proof. We give an fpt-nucomp-reduction from SATQUERY(query-size) to itself, where the resulting offline instance is in 3CNF. In order to do so, we need to specify fpt-size functions f_1, f_2, a computable function h and an fpt-time function g such that for each instance (φ, L, k) of SATQUERY(query-size) it holds that for each $m \geq |L|$, $(\varphi, L, k) \in$ SATQUERY(query-size) if and only if $(f_1(\varphi, 1^m, k), g(f_2(\varphi, 1^m, k), L, k), h(k)) \in$ SATQUERY(query-size). By using the standard Tseitin transformations [32], we can in polynomial time transform the formula φ into a 3CNF formula φ' with the property that $\text{Var}(\varphi) \subseteq \text{Var}(\varphi')$ and that for each truth assignment $\alpha : \text{Var}(\varphi) \to \{0,1\}$ it holds that $\varphi[\alpha]$ is true if and only if $\varphi'[\alpha]$ is satisfiable. We then let $f_1(\varphi, 1^m, k) = \varphi'$. Moreover, we let $f_2(\varphi, 1^m, k) = \emptyset$, we let $g(\emptyset, L, k) = L$, and we let $h(k) = k$. It is straightforward to verify the correctness of this fpt-nucomp-reduction. □

Proposition 90. *Let γ be a non-uniformly fpt-time computable generator. Then* $\text{FEWSAT}_\gamma \leq^{\text{fpt}}_{\text{nucomp}}$ SATQUERY(query-size).

Proof (sketch). We need to specify fpt-size functions f_1, f_2, a computable function h and an fpt-time function g such that for each instance (n, ℓ, k) of FEWSAT_γ it holds that for each $m \geq |\ell|$:

$$(n, \ell, k) \in \text{FEWSAT}_\gamma \qquad \text{if and only if} \qquad (f_1(n, 1^m, k), g(f_2(n, 1^m, k), \ell, k), h(k)) \in \text{SATQUERY(query-size)}.$$

We let $f_2(n, 1^m, k) = n$. In addition, we let $g(n, \ell, k)$ be an encoding L_ℓ of ℓ in terms of a set of literals of size k over the variables $x_{i,j}$, for $1 \leq i \leq n, 1 \leq j \leq k$. Then, we let $f_2(n, 1^m, k) = \varphi$ be a 3CNF formula that is satisfiable in conjunction with L_ℓ if and only if $\gamma(n, \ell, k)$ is satisfiable, for each $1 \leq \ell \leq n^k$. This can be done as follows.

By Proposition 69, we may assume without loss of generality that γ is a 3CNF generator. Firstly, the formula φ contains clauses to ensure that at most k variables $x_{i,j}$ are true. Then, we add $8n^3$ many variables y_i, corresponding to the $8n^3$ possible clauses $c_1, \ldots, c_{8n^3}$ of size 3 over the variables $x_1, \ldots, x_n$. Then, since for each $(n, k) \in \mathbb{N}^2$, the function $\gamma(n, \cdot, k)$ is computable in ftp-time, we can construct fpt-many clauses that ensure that, whenever L_ℓ is satisfied, the variables y_i must be set to true for the clauses $c_i \in \gamma(n, \ell, k)$. Finally, for each such possible clause c_i, we add clauses to ensure that whenever y_i is set to true, then the clause c_i must be satisfied. □

Proposition 91. *The class* nu-few-FPT *coincides with the set of all parameterized problems Q for which there exists a computable function f and a constant c such that for each $(n, k) \in \mathbb{N} \times \Sigma^*$ there is some circuit $\mathcal{C}_{n,k}$ of size $f(k)n^c$ such that, for each $\ell \in \{0,1\}^{k \log n}$, it holds that $(n, \ell, k) \in Q$ if and only if $\mathcal{C}_{n,k}[\ell]$ is satisfiable.*

Proof (idea). An argument similar to the one in the proof of Proposition 20 can be used to show this result. □

Proposition 92. *Let γ be a generator. Then* $\text{FEWSAT}_\gamma \in$ par-nucomp-FPT *if and only if* $\text{FEWSAT}_\gamma \in \text{FPT/fpt}$.

Proof. ($\Rightarrow$) Assume that there exists an fpt-size function f_1, an fpt-time function g a computable function h and a parameterized problem $Q \in \text{FPT}$ such that for each instance (n, ℓ, k) of FEWSAT_γ and each $m \geq |\ell|$ it holds that:

$$(n, \ell, k) \in \text{FEWSAT}_\gamma \qquad \text{if and only if} \qquad (f_1(n, 1^m, k), g(\ell, k), h(k)) \in Q.$$

We show that $\text{FEWSAT}_\gamma \in \text{FPT/fpt}$. Take an arbitrary $(n, k) \in \mathbb{N}^2$. Consider the string $\alpha = f_1(n, 1^m, k)$ as advice, for some $m \geq |\ell| = k \log n$. Then deciding for some $1 \leq \ell \leq n^k$ whether $(n, \ell, k) \in \text{FEWSAT}_\gamma$ can be done in fpt-time using the advice string α, by checking whether $(\alpha, g(\ell, k), h(k)) \in Q$.

($\Leftarrow$) Conversely, assume that $\text{FEWSAT}_\gamma \in \text{FPT/fpt}$, i.e., that there exist a computable function f and a constant c such that for each $(n, k) \in \mathbb{N}^2$ there exists some advice string $\alpha(n, k)$ of length $f(k)n^c$ such that, given $(\alpha(n, k), n, \ell, k)$, deciding whether $(n, \ell, k) \in \text{FEWSAT}_\gamma$ is fixed-parameter tractable. We then construct a fpt-size function f_1, an fpt-time function g and a computable function h such that there exists a parameterized problem $Q \in \text{FPT}$ such that for each instance (n, ℓ, k) of FEWSAT_γ and each $m \geq |\ell|$ it holds that:

$$(n, \ell, k) \in \text{FEWSAT}_\gamma \qquad \text{if and only if} \qquad (f_1(n, 1^m, k), g(\ell, k), h(k)) \in Q.$$

We let $f_1(n, 1^m, k) = (\alpha(n, k), n)$, we let $g(\ell, k) = \ell$ and we let $h(k) = k$. Then, by assumption, given $(f_1(n, 1^m, k), g(\ell, k), h(k))$, deciding whether $(n, \ell, k) \in \text{FEWSAT}_\gamma$ is fixed-parameter tractable. □

Proposition 93. *If* $\text{FEWSAT}_\gamma \in \text{FPT/fpt}$ *for each non-uniformly fpt-time computable generator* γ*, then* $\text{SATQUERY}(\text{query-size}) \in \text{par-nucomp-FPT}$.

Proof. In order to show that $\text{SATQUERY}(\text{query-size}) \in \text{par-nucomp-FPT}$, we specify an fpt-size function f_1, an fpt-time function g, a computable function h and a parameterized problem $Q \in \text{FPT}$ such that for all instances (φ, L, k) of $\text{SATQUERY}(\text{query-size})$ and each $m \geq |L| = k$ it holds that:

$$(\varphi, L, k) \in \text{SATQUERY}(\text{query-size}) \qquad \text{if and only if} \qquad (f_1(\varphi, 1^m, k), g(L, k), h(k)) \in Q.$$

Assume without loss of generality that φ contains the variables $x_1, \ldots, x_n$. There are (at most) $2n^k \leq n^{2k}$ many set of k literals over the variables $x_1, \ldots, x_n$. Let $L_1, \ldots, L_u$ be an enumeration of these clauses. We consider the following non-uniformly fpt-time computable generator γ_φ. Given a triple $(n', \ell, k') \in \mathbb{N}^3$, we define $\gamma_\varphi(n', \ell, k') = \emptyset$ if $n' \neq n$, or $k' \neq 2k$, or $\ell > u$; otherwise, we define $\gamma_\varphi(n', \ell, k') = \gamma_\varphi(n, \ell, 2k)$ to be a propositional formula φ' that is satisfiable if and only if $\varphi \wedge \bigwedge L_\ell$ is satisfiable. Then, $(n', \ell, k') \in \text{FEWSAT}_{\gamma_\varphi}$ if and only if $(\varphi, L_\ell, k) \in \text{SATQUERY}(\text{query-size})$. By assumption, $\text{FEWSAT}_{\gamma_\varphi} \in \text{FPT/fpt}$, and therefore there exists some algorithm $A_{(n',k')}$ that decides whether $(n', \ell, k') \in \text{FEWSAT}_{\gamma_\varphi}$ in fpt-time. We now let $f_1(\varphi, 1^m, k) = \alpha_{(n',k')}$ be a description of the algorithm $A_{(n',k')}$. Since $A_{(n',k')}$ runs in fixed-parameter tractable time, we know that its description $\alpha_{(n',k')}$ is of fpt-size. Moreover, we let $g(L, k) = \ell$, for $L = L_\ell$; and we let $h(k) = k' = 2k$. The problem Q consists of simulating the algorithm $A_{(n',k')}$ on input (n', ℓ, k'), which is fixed-parameter tractable. This completes our fpt-nucomp-reduction. □

The above results together give us the following characterization theorem.

Theorem 94. $\text{SATQUERY}(\text{query-size}) \in \text{par-nucomp-FPT}$ *if and only if* $\text{nu-few-NP} \subseteq \text{FPT/fpt}$.

Proof. ($\Rightarrow$) Assume that $\text{SATQUERY}(\text{query-size}) \in \text{par-nucomp-FPT}$. Then by Proposition 90, we know that $\text{FEWSAT}_\gamma \in \text{par-nucomp-FPT}$, for each non-uniformly fpt-time computable generator γ. Then, by Proposition 92, we know that $\text{FEWSAT}_\gamma \in \text{FPT/fpt}$, for each non-uniformly fpt-time computable generator γ. In other words, $\text{nu-few-NP} \subseteq \text{FPT/fpt}$.

($\Leftarrow$) Conversely, assume that $\text{nu-few-NP} \subseteq \text{FPT/fpt}$. Then for each non-uniformly fpt-time computable generator γ, it holds that $\text{FEWSAT}_\gamma \in \text{FPT/fpt}$. Then, by Proposition 93, we know that $\text{SATQUERY}(\text{query-size}) \in \text{par-nucomp-FPT}$. □

With this characterization of the parameterized compilability of $\text{SATQUERY}(\text{query-size})$, we can return to the problem $\text{CONSTRAINED-CLIQUE}(\text{constr.-size})$.

Proposition 95. $\text{CONSTRAINED-CLIQUE}(\text{constr.-size})$ *is equivalent to* $\text{SATQUERY}(\text{size})$ *under fpt-nucomp-reductions.*

Proof. Firstly, we give an fpt-nucomp-reduction from $\text{SATQUERY}(\text{query-size})$ to $\text{CONSTRAINED-CLIQUE}(\text{constr.-size})$. Before we specify this reduction, we will look at a well-known polynomial-time reduction from 3SAT to CLIQUE. Let $\varphi = c_1 \wedge \cdots \wedge c_b$, and let $\text{Var}(\varphi) = \{x_1, \ldots, x_n\}$. We construct a graph $G = (V, E)$ as follows. We let $V = \{ (c_j, l) : 1 \leq j \leq b, l \in c_j \} \cup \{ x_i, \overline{x_i} : 1 \leq i \leq n \}$. We describe the edges of G as three sets: (1) the edges between two vertices of the form (c_j, l), (2) the edges between two vertices of the form $x_i, \overline{x_i}$, and (3) the edges between a vertex of the form (c_j, l) and a vertex of the form $x_i, \overline{x_i}$. We describe set (1). Let (c_j, l) and $(c_{j'}, l')$ be two vertices. These vertices are connected by an edge if and only if both $j \neq j'$ and $l \neq \overline{l'}$. Next, we describe set (2). Two vertices $v_1, v_2 \in \{ x_i, \overline{x_i} : 1 \leq i \leq n \}$ are connected by an edge if and only if v_1 and v_2 are not complementary literals x_i and $\overline{x_i}$, for some $1 \leq i \leq n$. Finally, we describe set (3). Let (c_j, l) and $v \in \{ x_i, \overline{x_i} : 1 \leq i \leq n \}$ be two vertices. Then (c_j, l) and v are connected by an edge if and only if l and v are not complementary literals x_i and $\overline{x_i}$, for some $1 \leq i \leq n$. We have that G has a clique of size $n + b$ if and only if φ is satisfiable. Moreover, each satisfying assignment $\alpha : \{x_1, \ldots, x_n\} \to \{0, 1\}$ of φ corresponds to a clique $C = \{ l \in \{ x_i, \overline{x_i} : 1 \leq i \leq n \} : \alpha(l) = 1 \} \cup D$ of size $n + b$, for some $D \subseteq \{ (c_j, l) \in V : \alpha(l) = 1 \}$, and vice versa. This direct correspondence between cliques and satisfying truth assignments has the following consequence. Let L be a set of literals. There exists a clique of size $n + b$ in G that does not contain the vertices $l \in L$ if and only if there exists a satisfying truth assignment of φ

that does not satisfy any literal $l \in L$, which is the case if and only if there exists a satisfying truth assignment of φ that satisfies the complement $\bar{l}$ of each $l \in L$.

We are now ready to specify our fpt-nucomp-reduction. In order to do so, we need to specify fpt-size functions f_1, f_2, a computable function h and an fpt-time function g. Let (φ, L, k) be an instance of SATQUERY(query-size). We may assume without loss of generality that φ is in 3CNF. We let $f_1(\varphi, 1^m, k) = G$, where G is the graph constructed from φ as in the polynomial-time reduction from 3SAT to CLIQUE that is described above. We let $f_2(\varphi, 1^m, k) = u = n + b$; we let $g(u, L, k) = (u, V_1, \emptyset)$, where $V_1 = \{\, \bar{l} : l \in L \,\}$ is the set of vertices $\bar{l}$ in V corresponding to the complements of the literals in L. Finally, we let $h(k) = k$. As argued above, we have that $\varphi \wedge \bigwedge_{l \in L} l$ is satisfiable if and only if G has a clique of size u that does not contain any vertex in V_1. Therefore, this reduction is correct.

Next, we give an fpt-nucomp-reduction from CONSTRAINED-CLIQUE(constr.-size) to SATQUERY(query-size). In order to do so, we consider a polynomial-time reduction from CLIQUE to 3SAT. It is straightforward to construct, given a graph $G = (V, E)$ and an integer $1 \leq u \leq |V|$, a 3CNF formula φ_u in polynomial time, such that φ_u is satisfiable if and only if G contains a clique of size u. Moreover, we can do this in such a way that the formulas $\varphi_1, \ldots, \varphi_{|V|}$ share a set of variables $\{\, x_v : v \in V \,\}$ with the property that for each $1 \leq u \leq |V|$ and each two subset $V_1, V_2 \subseteq V$ it holds that G has a clique $C \subseteq V$ of size u that contains no vertex in V_1 and that contains all vertices in V_2 if and only if the formula $\varphi_u \wedge \bigwedge l \in L$ is satisfiable, where $L = \{\, \overline{x_v} : v \in V_1 \,\} \cup \{\, x_v : v \in V_2 \,\}$.

Now, we are ready to specify our fpt-nucomp-reduction. In order to do so, we need to specify fpt-size functions f_1, f_2, a computable function h and an fpt-time function g. Let $(G, (V_1, V_2, u), k)$ be an instance of CONSTRAINED-CLIQUE(constr.-size). We let $f_1(G, 1^m, k) = \psi$, where we define $\psi = \bigwedge_{1 \leq j \leq |V|} (y_j \rightarrow \varphi_j)$, where the formulas φ_j are constructed as in the polynomial-time reduction from CLIQUE to 3SAT that is described above. We let $f_2(G, 1^m, k) = \emptyset$, we let $g(\emptyset, (V_1, V_2, u), k) = L \cup \{y_u\}$, where $L = \{\, \overline{x_v} : v \in V_1 \,\} \cup \{\, x_v : v \in V_2 \,\}$, and we let $h(k) = k + 1$. As argued above, we have that G has a clique of size u that does not contain any vertex in V_1 and that contains all vertices in V_2 if and only if $\varphi \wedge \bigwedge_{l \in L} l \wedge y_u$ is satisfiable. Therefore, this reduction is correct. □

Corollary 96. CONSTRAINED-CLIQUE(constr.-size) $\in$ par-comp-FPT *implies that* nu-few-NP $\subseteq$ FPT/fpt.

4.4.1 Clique size as part of the offline instance

In fact, requiring that the size of the cliques is part of the offline instance (rather than part of the online instance) does not make a difference for the compilability of the problem. Consider the following parameterized compilation problem, where the size of the cliques is part of the offline instance.

CONSTRAINED-CLIQUE$^{\text{offline-size}}$(constr.-size)
Offline instance: a graph $G = (V, E)$, and a positive integer $u \geq 1$.
Online instance: two subset $V_1, V_2 \subseteq V$ of vertices.
Parameter: $k = |V_1| + |V_2|$.
Question: is there a clique $C \subseteq V$ in G of size u such that $C \cap V_1 = \emptyset$ and $V_2 \subseteq C$?

Proposition 97. CONSTRAINED-CLIQUE$^{\text{offline-size}}$(constr.-size) *is equivalent to* SATQUERY(query-size) *under fpt-nucomp-reductions.*

Proof. The proof of Proposition 95 can be easily modified to show this result. □

4.5 Other parameterized compilation problems

We consider some other parameterized compilation problems that bear a similar relation to a collapse in the non-uniform complexity landscape as CONSTRAINED-CLIQUE(constr.-size).

4.5.1 Hamiltonian Paths and the Travelling Salesman Problem

Using ideas similar to the ones used in the above section with compilation problems related to cliques, we can show for some parameterized compilation problems related to finding Hamiltonian paths and related to the Travelling Salesperson Problem that they are equivalent to SATQUERY(query-size) (under fpt-nucomp-reductions). The main idea

that we use to show these equivalences is that the standard polynomial-time reductions from 3SAT to finding Hamiltonian paths in (undirected or directed) graphs or to the TSP result in instances where specific edges in the solutions correspond to the assignment of specific literals to specific truth values in satisfying assignments of the original 3CNF formula. We consider the following parameterized compilation problems.

CONSTRAINED-HP-UNDIRECTED(constr.-size)
Offline instance: an undirected graph $G = (V, E)$.
Online instance: two subsets $E_1, E_2 \subseteq E$ of edges.
Parameter: $k = |E_1| + |E_2|$.
Question: is there a Hamiltonian path π in G such that π includes no edge in E_1 and includes each edge in E_2?

CONSTRAINED-HP-DIRECTED(constr.-size)
Offline instance: a directed graph $G = (V, E)$.
Online instance: two subsets $E_1, E_2 \subseteq E$ of edges.
Parameter: $k = |E_1| + |E_2|$.
Question: is there a Hamiltonian path π in G such that π includes no edge in E_1 and includes each edge in E_2?

CONSTRAINED-TSP(constr.-size)
Offline instance: a directed graph $G = (V, E)$, and a cost $c(e) \in \mathbb{N}$ (given in binary) for each edge $e \in E$.
Online instance: two subsets $E_1, E_2 \subseteq E$ of edges, and a positive integer $u \geq 1$ (given in binary).
Parameter: $k = |E_1| + |E_2|$.
Question: is there a Hamiltonian cycle π in G of total cost $\leq u$ such that π includes no edge in E_1 and includes each edge in E_2?

Proposition 98. *The following three parameterized compilation problems are equivalent to* SATQUERY(query-size) *(under fpt-nucomp-reductions):*

- CONSTRAINED-HP-UNDIRECTED(constr.-size)*,*
- CONSTRAINED-HP-DIRECTED(constr.-size)*, and*
- CONSTRAINED-TSP(constr.-size).

Proof (sketch). The problems of finding a Hamiltonian path (in an undirected or in a directed graph), finding a Hamiltonian cycle (in an undirected or in a directed graph), and finding a tour of length $\leq u$ for an instance of the TSP are classic NP-complete problems. The well-known polynomial-time reductions from these problems to 3SAT and from 3SAT to these problems have a direct correspondence between solutions of these problems and satisfying assignments of the 3SAT instances, similar to the correspondence used in the proof of Proposition 95. Using arguments that are completely analogous to the arguments used in these proofs, we can show that the problems CONSTRAINED-HP-UNDIRECTED(constr.-size), CONSTRAINED-HP-DIRECTED(constr.-size), and CONSTRAINED-TSP(constr.-size) are equivalent under fpt-nucomp-reductions to SATQUERY(query-size).

Note that the standard reductions can be modified straightforwardly to work also for the restrictions of CONSTRAINED-HP-UNDIRECTED(constr.-size), CONSTRAINED-HP-DIRECTED(constr.-size), and CONSTRAINED-TSP(constr.-size), where $E_1 = \emptyset$, as well as for the restriction where $E_2 = \emptyset$. □

4.5.2 Graph Colorability

Next, we look at parameterized compilation problems that are based on graph colorability problems. Again, by exploiting the well-known polynomial-time reductions to and from 3SAT, and the direct correspondence between solution to the coloring problems and the satisfiability problem, we can establish equivalence between the following problems and SATQUERY(query-size). We omit the straightforward proofs. In addition, we can derive classical (in)compilability results hold for unparameterized variants of the following problems, similarly to the cases in the above sections.

CONSTRAINED-3COL(constr.-size)
Offline instance: a graph $G = (V, E)$.
Online instance: a partial 3-coloring ρ, that assigns a color in $\{1, 2, 3\}$ to a subset $V' \subseteq V$ of vertices.
Parameter: $k = |V'|$.
Question: is there a proper 3-coloring ρ' of G that extends ρ?

Let $r \geq 1$ be a fixed positive integer.

CONSTRAINED-rCOL(constr.-size)
Offline instance: a graph $G = (V, E)$.
Online instance: a partial r-coloring ρ, that assigns a color in $\{1, 2, 3\}$ to a subset $V' \subseteq V$ of vertices.
Parameter: $k = |V'|$.
Question: is there a proper r-coloring ρ' of G that extends ρ?

Proposition 99. *The following parameterized compilation problems are equivalent to* SATQUERY(query-size) *(under fpt-nucomp-reductions):*

- CONSTRAINED-3COL(constr.-size)*, and*
- CONSTRAINED-rCOL(constr.-size).

4.5.3 Other Problems

Again, using similar ideas, exploiting well-known polynomial time reductions to and from 3SAT, we observe that the following parameterized compilation problems are equivalent to SATQUERY(query-size) (under fpt-nucomp-reductions). We omit the straightforward proofs.

CONSTRAINED-IS(constr.-size)
Offline instance: a graph $G = (V, E)$.
Online instance: two subsets $V_1, V_2 \subseteq V$ of vertices, and a positive integer $u \geq 1$.
Parameter: $k = |V_1| + |V_2|$.
Question: is there an independent set $C \subseteq V$ in G of size $\geq u$ such that $C \cap V_1 = \emptyset$ and $V_2 \subseteq C$?

CONSTRAINED-VC(constr.-size)
Offline instance: a graph $G = (V, E)$.
Online instance: two subsets $V_1, V_2 \subseteq V$ of vertices, and a positive integer $u \geq 1$.
Parameter: $k = |V_1| + |V_2|$.
Question: is there a vertex cover $C \subseteq V$ of G of size $\leq u$ such that $C \cap V_1 = \emptyset$ and $V_2 \subseteq C$?

CONSTRAINED-DS(constr.-size)
Offline instance: a graph $G = (V, E)$.
Online instance: two subsets $V_1, V_2 \subseteq V$ of vertices, and a positive integer $u \geq 1$.
Parameter: $k = |V_1| + |V_2|$.
Question: is there a dominating set $C \subseteq V$ of G of size $\leq u$ such that $C \cap V_1 = \emptyset$ and $V_2 \subseteq C$?

CONSTRAINED-HS(constr.-size)
Offline instance: a finite set T, and a collection $\mathcal{S} = \{S_1, \ldots, S_b\}$ of subsets of T.
Online instance: two subsets $T_1, T_2 \subseteq T$ of elements, and a positive integer $u \geq 1$.
Parameter: $k = |T_1| + |T_2|$.
Question: is there a hitting set $H \subseteq T$ of $\mathcal{S}$ of size $\leq u$ such that $H \cap T_1 = \emptyset$ and $T_2 \subseteq H$?

A *kernel* in a directed graph $G = (V, E)$ is a set $K \subseteq V$ of vertices such that (1) no two vertices in K are adjacent, and (2) for every vertex $u \in V \backslash K$ there is a vertex $v \in K$ such that $(u, v) \in E$.

CONSTRAINED-KERNEL(constr.-size)
Offline instance: a directed graph $G = (V, E)$.
Online instance: two subsets $V_1, V_2 \subseteq V$ of vertices.
Parameter: $k = |V_1| + |V_2|$.
Question: is there a kernel $K \subseteq V$ of G such that $K \cap V_1 = \emptyset$ and $V_2 \subseteq C$?

CONSTRAINED-EC(constr.-size)
Offline instance: a finite set T, and a collection $\mathcal{S} = \{S_1, \ldots, S_b\}$ of subsets of T.
Online instance: two subsets $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{S}$.
Parameter: $k = |\mathcal{S}_1| + |\mathcal{S}_2|$.
Question: is there an exact cover $C \subseteq \mathcal{S}$ of T, i.e., a set C such that $\bigcup C = T$ and for each $S, S' \in C$ with $S \neq S'$ it holds that $S \cap S' = \emptyset$, such that $C \cap \mathcal{S}_1 = \emptyset$ and $\mathcal{S}_2 \subseteq C$?

CONSTRAINED-EC3S(constr.-size)
Offline instance: a finite set T, and a collection $\mathcal{S} = \{S_1, \ldots, S_b\}$ of subsets of T, each of size 3.
Online instance: two subsets $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{S}$.
Parameter: $k = |\mathcal{S}_1| + |\mathcal{S}_2|$.
Question: is there an exact cover $C \subseteq \mathcal{S}$ of T, i.e., a set C such that $\bigcup C = T$ and for each $S, S' \in C$ with $S \neq S'$ it holds that $S \cap S' = \emptyset$, such that $C \cap \mathcal{S}_1 = \emptyset$ and $\mathcal{S}_2 \subseteq C$?

CONSTRAINED-KNAPSACK(constr.-size)
Offline instance: a finite set T of elements, and for each $t \in T$ a cost $c(t) \in \mathbb{N}$ and a value $v(t) \in \mathbb{N}$, both given in binary.
Online instance: two subsets $T_1, T_2 \subseteq T$, and two positive integers $u_1, u_2 \geq 1$, both given in binary.
Parameter: $k = |T_1| + |T_2|$.
Question: is there an subset $S \subseteq T$ such that $\sum_{s \in S} c(s) \leq u_1$, such that $\sum_{s \in S} v(s) \geq u_2$, and such that $S \cap T_1 = \emptyset$ and $T_2 \subseteq S$?

Proposition 100. *The following parameterized compilation problems are equivalent to* SATQUERY(query-size) *(under fpt-nucomp-reductions):*

- CONSTRAINED-IS(constr.-size),
- CONSTRAINED-VC(constr.-size),
- CONSTRAINED-DS(constr.-size),
- CONSTRAINED-HS(constr.-size),
- CONSTRAINED-KERNEL(constr.-size),
- CONSTRAINED-EC(constr.-size),
- CONSTRAINED-EC3S(constr.-size), *and*
- CONSTRAINED-KNAPSACK(constr.-size).

5 Relation to (a Parameterized Variant of) the Polynomial Hierarchy

We give an overview of the relation between non-uniform parameterized complexity classes and a parameterized variant of the Polynomial Hierarchy (PH). We begin with reviewing the PH and its parameterized variant [15]. Then, we show that certain inclusions of non-uniform parameterized complexity classes entail collapses in the parameterized variant of the PH.

The Polynomial Hierarchy There are many natural decision problems that are commonly assumed not to be contained in the classical complexity classes P and NP (under some common complexity-theoretic assumptions). The *Polynomial Hierarchy* [23, 27, 31, 33] contains a hierarchy of increasing complexity classes Σ_i^{p}, for all $i \geq 0$. We give a characterization of these classes based on the satisfiability problem of various classes of quantified Boolean formulas. A *quantified Boolean formula* (in prenex form) is a formula of the form $Q_1 X_1 Q_2 X_2 \ldots Q_m X_m \psi$, where each Q_i is either $\forall$ or $\exists$, the X_i are disjoint sets of propositional variables, and ψ is a Boolean formula over the variables in $\bigcup_{i=1}^{m} X_i$. The quantifier-free part of such formulas is called the *matrix* of the formula. Truth of such formulas is defined in the usual way. Let $\gamma = \{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$ be a function that maps some variables of a formula φ to other variables or to truth values. We let $\varphi[\gamma]$ denote the application of such a substitution γ to the formula φ. We also write $\varphi[x_1 \mapsto d_1, \ldots, x_n \mapsto d_n]$ to denote $\varphi[\gamma]$. For each $i \geq 1$ we define the following decision problem.

QSAT_i
Instance: A quantified Boolean formula $\varphi = \exists X_1 \forall X_2 \exists X_3 \ldots Q_i X_i \psi$, where Q_i is a universal quantifier if i is even and an existential quantifier if i is odd.
Question: Is φ true?

Input formulas to the problem QSAT_i are called Σ_i^{p}-formulas. For each nonnegative integer $i \geq 0$, the complexity class Σ_i^{p} can be characterized as the closure of the problem QSAT_i under polynomial-time reductions [31, 33]. The Σ_i^{p}-hardness of QSAT_i holds already when the matrix of the input formula is restricted to 3CNF for odd i, and restricted to 3DNF for even i. Note that the class Σ_0^{p} coincides with P, and the class Σ_1^{p} coincides with NP. For each $i \geq 1$, the class Π_i^{p} is defined as co-Σ_i^{p}.

The classes Σ_i^{p} and Π_i^{p} can also be defined by means of nondeterministic Turing machines with an oracle. For any complexity class C, we let NP^C be the set of decision problems that is decided in polynomial time by a nondeterministic Turing machine with an oracle for a problem in C. Then, the classes Σ_i^{p} and Π_i^{p}, for $i \geq 0$, can be equivalently defined by letting $\Sigma_0^{\mathrm{p}} = \Pi_0^{\mathrm{p}} = \mathrm{P}$, and for each $i \geq 1$ letting $\Sigma_i^{\mathrm{p}} = \mathrm{NP}^{\Sigma_{i-1}^{\mathrm{p}}}$ and $\Pi_i^{\mathrm{p}} = \text{co-NP}^{\Sigma_{i-1}^{\mathrm{p}}}$.

A parameterized variant of the Polynomial Hierarchy Problems in NP and co-NP can be encoded into SAT in such a way that the time required to produce the encoding and consequently also the size of the resulting SAT instance are polynomial in the input (the encoding is a polynomial-time many-one reduction). Typically, the SAT encodings of problems proposed for practical use are of this kind (cf. [29]). For problems that are "beyond NP," say for problems on the second level of the PH, such polynomial SAT encodings do not exist, unless the PH collapses. However, for such problems, there still could exist SAT encodings which can be produced in fpt-time in terms of some parameter associated with the problem. In fact, such fpt-time SAT encodings have been obtained for various problems that lie on the second level of the PH or that are even harder [10, 12, 16, 28]. The classes para-NP and para-co-NP contain exactly those parameterized problems that admit such a many-one fpt-reduction to SAT and UNSAT, respectively. Thus, with fpt-time encodings, one can go significantly beyond what is possible by conventional polynomial-time SAT encodings.

Fpt-time encodings to SAT also have their limits. Clearly, para-Σ_2^{p}-hard and para-Π_2^{p}-hard parameterized problems do not admit fpt-time encodings to SAT, even when the parameter is a constant, unless the PH collapses. There are problems that apparently do not admit fpt-time encodings to SAT, but seem not to be para-Σ_2^{p}-hard nor para-Π_2^{p}-hard either. Recently, several complexity classes have been introduced to classify such intermediate problems [16, 15]. These parameterized complexity classes are dubbed the k-$*$ class and the $*$-k hierarchy, inspired by their definition, which is based on the following weighted variants of the quantified Boolean satisfiability problem that is canonical for the second level of the PH. Let $\mathcal{C}$ be a class of Boolean circuits. The problem $\exists^k \forall^*$-$\mathrm{WSAT}(\mathcal{C})$ provides the foundation for the k-$*$ class.

$\exists^k \forall^*$-WSAT
Instance: A quantified Boolean formula $\exists X . \forall Y . \psi$, and an integer k.
Parameter: k.
Question: Does there exist a truth assignment α to X with weight k such that for all truth assignments β to Y the assignment $\alpha \cup \beta$ satisfies ψ?

Similarly, the problem $\exists^* \forall^k$-$\mathrm{WSAT}(\mathcal{C})$ provides the foundation for the $*$-k hierarchy.

$\exists^*\forall^k$-WSAT($\mathcal{C}$)
Instance: A Boolean circuit $C \in \mathcal{C}$ over two disjoint sets X and Y of variables, and an integer k.
Parameter: k.
Question: Does there exist a truth assignment α to X such that for all truth assignments β to Y with weight k the assignment $\alpha \cup \beta$ satisfies C?

The parameterized complexity class $\exists^k\forall^*$ (also called the k-$*$ class) is then defined as follows:

$$\exists^k\forall^* = \quad [\ \exists^k\forall^*\text{-WSAT}\ ]_{\text{fpt}}.$$

Similarly, the classes of the $*$-k hierarchy are defined as follows. Remember that we denote the class of Boolean circuits with depth u and weft t by $\text{CIRC}_{t,u}$, the class of all Boolean circuits by CIRC and the class of all Boolean formulas by FORM.

$$\begin{aligned}
\exists^*\forall^k\text{-W[t]} &= \quad [\ \{\ \exists^*\forall^k\text{-WSAT}(\text{CIRC}_{t,u}) : u \geq 1\ \}\ ]_{\text{fpt}},\ \text{for each t} \geq 1;\\
\exists^*\forall^k\text{-W[SAT]} &= \quad [\ \exists^*\forall^k\text{-WSAT}(\text{FORM})\ ]_{\text{fpt}};\ \text{and}\\
\exists^*\forall^k\text{-W[P]} &= \quad [\ \exists^*\forall^k\text{-WSAT}(\text{CIRC})\ ]_{\text{fpt}}.
\end{aligned}$$

Note that these definitions are entirely analogous to those of the parameterized complexity classes W[t] of the Weft hierarchy [8]. The following inclusion relations hold between the classes of the $*$-k hierarchy:

$$\exists^*\forall^k\text{-W[1]} \subseteq \exists^*\forall^k\text{-W[2]} \subseteq \cdots \subseteq \exists^*\forall^k\text{-W[SAT]} \subseteq \exists^*\forall^k\text{-W[P]}.$$

Dual to the classical complexity class Σ_2^{p} is its co-class Π_2^{p}, whose canonical complete problem is complementary to the problem QSAT_2. Similarly, we can define dual classes for the k-$*$ class and for each of the parameterized complexity classes in the $*$-k hierarchy. These co-classes are based on problems complementary to the problems $\exists^k\forall^*$-WSAT and $\exists^*\forall^k$-WSAT, i.e., these problems have as yes-instances exactly the no-instances of $\exists^k\forall^*$-WSAT and $\exists^*\forall^k$-WSAT, respectively. Equivalently, these complementary problems can be considered as variants of $\exists^k\forall^*$-WSAT and $\exists^*\forall^k$-WSAT where the existential and universal quantifiers are swapped, and are therefore denoted with $\forall^k\exists^*$-WSAT and $\forall^*\exists^k$-WSAT. We use a similar notation for the dual complexity classes, e.g., we denote co-$\exists^*\forall^k$-W[t] by $\forall^*\exists^k$-W[t].

Parameterized Variants of the Karp-Lipton Theorem Next, we move to results relating inclusions between certain non-uniform parameterized complexity classes to collapses in the parameterized variant of the PH. The following results are analogues of the Karp-Lipton Theorem related to non-uniform parameterized complexity and parameterized complexity at higher levels of the PH.

In order to develop these results, we consider another non-uniform variant of parameterized complexity classes, based on advice of size $f(k)\log n$ (we call this *log-kernel-size advice*).

Definition 101 (*log-kernel-size advice*)**.** *Let C be a parameterized complexity class. We define C/*log-kernel *to be the class of all parameterized problems Q for which there exists a parameterized problem $Q' \in C$ and a computable function f such that for each $(n,k) \in \mathbb{N} \times \Sigma^*$ there exists some $\alpha(n,k) \in \Sigma^*$ of size $f(k)\log n$ with the property that for all instances (x,k) with $|x| = n$, it holds that $(x,k) \in Q$ if and only if $(x, \alpha(|x|,k),k) \in Q'$.*

Proposition 102. *If* W[P] $\subseteq$ FPT/log-kernel*, then* $\forall^*\exists^k$-W[P] $\subseteq \exists^k\forall^*$.

Proof. Suppose that W[P] $\subseteq$ FPT/log-kernel. Then there exists a computable function f such that for each $(n,k) \in \mathbb{N} \times \mathbb{N}$ there is some advice string $\alpha(n,k)$ of length $f(k)\log n$ such that for any instance (x,k) of WSAT(CIRC) with $|x| = n$, deciding whether $(x,k) \in$ WSAT(CIRC), given $(x,\alpha(n,k),k)$ is fixed-parameter tractable. Then, by self-reducibility of WSAT(CIRC), we can straightforwardly transform this fpt-algorithm A_1 that decides whether $(x,k) \in$ WSAT(CIRC) into an fpt-algorithm A_2 that, given $(x,\alpha(n,k),k)$, computes a satisfying assignment for x of weight k, if it exists, and fails otherwise. Here we assume without loss of generality that instances of WSAT(CIRC) are encoded in such a way that we can instantiate variables to truth values without changing the size of the instance.

Our assumption that W[P] $\subseteq$ FPT/log-kernel only implies that for each (n, k), a suitable advice string $\alpha(n, k)$ exists. The idea of this proof is that we can guess this advice string using (bounded weight) existential quantification. For any guessed string $\alpha(n, k)$, we can use it to execute the fpt-algorithm A_2 to compute a satisfying assignment (of weight k) for an instance of $\forall^*\exists^k$-WSAT(CIRC) and to verify that this is in fact a correct satisfying assignment. Moreover, we know that (at least) one of the advice strings $\alpha(n, k)$ is correct, so if no guess leads to a satisfying assignment of weigth k, we can conclude that no such assignment exists.

We show that the $\forall^*\exists^k$-W[P]-complete problem $\forall^*\exists^k$-WSAT(CIRC) is contained in $\exists^k\forall^*$. This suffices to show that $\forall^*\exists^k$-W[P] $\subseteq \exists^k\forall^*$. We do so by giving an algorithm that solves $\forall^*\exists^k$-WSAT(CIRC) and that can be implemented by an $\exists^k\forall^*$-machine [15, 17], i.e., an algorithm that can use $f(k)\log n$ existential nondeterministic steps, followed by $f(k)n^c$ universal nondeterministic steps, where n is the input size, f is some computable function and c is some constant.

Take an arbitrary instance of $\forall^*\exists^k$-WSAT(CIRC) consisting of the quantified Boolean instance $\forall X.\exists Y.C(X, Y)$ and the positive integer k. Let n denote the size of this instance. We construct the following algorithm B. Firstly, B guesses an advice string $\alpha(n, k) \in \{0, 1\}^{f(k)\log n}$. Since we have an upper bound of $f(k)\log n$ on the length of this string, we can guess this string in the $f(k)\log n$ existential nondeterministic steps of the algorithm. Next, the algorithm needs to verify that for all truth assignments γ (of any weight) to the variables in X, the instance $C(X, Y)[\gamma]$ of WSAT(CIRC) is a yes-instance. Using universal nondeterministic steps, the algorithm B guesses a truth assignment γ to the variables in X. We assume without loss of generality that for any γ the instance $C(X, Y)[\gamma]$ is of size n; if this instance is smaller, we can simply add some padding to ensure that it is of the right size. Then, using the guessed string $\alpha(n, k)$, the algorithm simulates the (deterministic) fpt-algorithm A_2 to decide whether $(C(X, Y)[\gamma], k) \in$ WSAT(CIRC). Moreover, it verifies whether the truth assignment of weight k that is constructed by A_2 (if any) in fact satisfies $C(X, Y)[\gamma]$. Clearly, this can be done using universal nondeterministic steps, since deterministic steps are a special case of universal nondeterministic steps. Finally, the algorithm accepts if and only if $C(X, Y)[\gamma]$ has a satisfying assignment of weight k.

To see that the algorithm B correctly decides $\forall^*\exists^k$-WSAT(CIRC), we observe the following. If the algorithm accepts an instance (C, k), then it must be the case that for all truth assignments γ to the universal variables of C the algorithm verified that there exists a satisfying assignment of weight k for $C[\gamma]$. Therefore, $(C, k) \in \forall^*\exists^k$-WSAT(CIRC). Conversely, if B rejects an instance (C, k), it means that for all advice strings $\alpha(n, k)$ of length $f(k)\log n$, there is some truth assignment γ to the universal variables of C such that simulating A_2 with $\alpha(n, k)$ does not yield a satisfying assignment. Then, since by assumption, at least one such advice string $\alpha(n, k)$ leads to correct behavior of A_2, we get that there exists some truth assignment γ to the universal variables of C such that $C[\gamma]$ does not have a satisfying assignment of weight k. In other words, $(C, k) \notin \forall^*\exists^k$-WSAT(CIRC). □

Corollary 103. W[P] $\not\subseteq$ FPT/kernel *and* W[P] $\not\subseteq$ FPT/slice*, unless* $\forall^*\exists^k$-W[P] $\subseteq \exists^k\forall^*$.

Proposition 104. *For each* t ≥ 1*, it holds that* W[t] $\subseteq$ FPT/log-kernel *implies* $\forall^*\exists^k$-W[t] $\subseteq \exists^k\forall^*$. *Also,* W[SAT] $\subseteq$ FPT/log-kernel *implies* $\forall^*\exists^k$-W[SAT] $\subseteq \exists^k\forall^*$

Proof. The proof of Proposition 102 can straightforwardly be modified to show this result. □

Corollary 105. *For each* t ≥ 1*, it holds that* W[1] $\not\subseteq$ FPT/kernel *and* W[1] $\not\subseteq$ FPT/slice*, unless* $\forall^*\exists^k$-W[1] $\subseteq \exists^k\forall^*$.

Corollary 106. *It holds that* W[SAT] $\not\subseteq$ FPT/kernel *and* W[SAT] $\not\subseteq$ FPT/slice*, unless* $\forall^*\exists^k$-W[SAT] $\subseteq \exists^k\forall^*$.

In addition, a straightforward modification of the proof of Proposition 102 can be used to relate the inclusions W[t] $\subseteq$ FPT/fpt to the parameterized variant of the PH.

Proposition 107. *For each* t $\in \{ n \geq 1 : n \in \mathbb{N} \} \cup \{\text{SAT}, \text{P}\}$ *it holds that* W[t] $\not\subseteq$ FPT/fpt*, unless* $\forall^*\exists^k$-W[t] $\subseteq$ para-Σ_2^{p}.

Proof. The proof of Proposition 102 can straightforwardly be modified to show this result. □

6 Relation to Probabilistic Parameterized Complexity

In this section, we briefly discuss the relation between probabilistic (and randomized) parameterized complexity and non-uniform parameterized complexity classes.

The class BPFPT [24] is one parameterized analogue of BPP, which is contained in XP. However, one can also consider the class para-BPP (which has also been called para-NP-BPFPT in the literature [25]). We have that BPFPT $\subseteq$ para-BPP [25]. In analogy to the inclusion BPP $\subseteq$ P/poly, we get that para-BPP $\subseteq$ FPT/fpt, but para-BPP $\not\subseteq$ FPT/slice unless BPP = P.

Proposition 108. para-BPP $\subseteq$ FPT/fpt.

Proof (sketch). Entirely analogous to the proof that BPP $\subseteq$ P/poly (cf. [2, Theorem 7.14]). □

Proposition 109. para-BPP $\not\subseteq$ FPT/slice *unless* BPP = P.

Proof. Suppose that para-BPP $\subseteq$ FPT/slice. We show that BPP $\subseteq$ P. Let $Q \in$ BPP be an arbitrary problem. Then consider the parameterized problem $Q' = \{ (x, 1) : x \in Q \}$. Clearly, $Q' \in$ para-BPP. By assumption, then also $Q' \in$ FPT/slice. Then the slice Q'_1 of Q' where the parameter value is 1 is polynomial-time solvable. From this, it easily follows that $Q \in$ P. Thus, we can conclude that BPP $\subseteq$ P. □

Corollary 110. para-BPP $\subseteq$ FPT/slice *if and only if* BPP = P.

7 Conclusion

We gave an overview of different non-uniform variants of parameterized complexity classes, in a homogeneous notation allowing a clear comparison of the different variants. We provided some (inclusion and separation) results relating the different non-uniform classes to each other. Additionally, we discussed the relation between parameterized knowledge compilation and non-uniform parameterized complexity.

Future work includes investigating the connections between the various non-uniform parameterized complexity classes and the compilability framework by Chen [6] that is phrased in terms of parameterized complexity classes that are contained in FPT.

References

[1] Eric W. Allender and Roy S. Rubinstein. P-printable sets. *SIAM J. Comput.*, 17(6):1193–1202, 1988.

[2] Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.

[3] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. Preprocessing of intractable problems. *Information and Computation*, 176(2):89–120, 2002.

[4] Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997.

[5] Hubie Chen. Parameterized compilability. In *Proceedings of IJCAI 2005, the Nineteenth International Joint Conference on Artificial Intelligence*, 2005.

[6] Hubie Chen. Parameter compilation. *CoRR*, abs/1503.00260, 2015.

[7] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annual Symp. on Theory of Computing (STOC 1971)*, pages 151–158, Shaker Heights, Ohio, 1971.

[8] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer Verlag, New York, 1999.

[9] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013.

[10] Ulle Endriss, Ronald de Haan, and Stefan Szeider. Parameterized complexity results for agenda safety in judgment aggregation. In *Proceedings of the 5th International Workshop on Computational Social Choice (COMSOC-2014)*. Carnegie Mellon University, June 2014.

[11] Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

[12] Johannes Klaus Fichte and Stefan Szeider. Backdoors to normality for disjunctive logic programs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*, pages 320–327. AAAI Press, 2013.

[13] Jörg Flum and Martin Grohe. Describing parameterized complexity classes. *Information and Computation*, 187(2):291–319, 2003.

[14] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin, 2006.

[15] Ronald de Haan and Stefan Szeider. The parameterized complexity of reasoning problems beyond NP. Technical Report 1312.1672v3, arXiv.org, 2014.

[16] Ronald de Haan and Stefan Szeider. The parameterized complexity of reasoning problems beyond NP. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Proceedings of the Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2014), Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014.

[17] Ronald de Haan and Stefan Szeider. Machine characterizations for parameterized complexity classes beyond para-NP. In *Proceedings of the 41st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2015)*, Lecture Notes in Computer Science. Springer Verlag, 2015. To appear.

[18] Juris Hartmanis and Richard E. Stearns. On the computational complexity of algorithms. *American Mathematical Society*, pages 285–306, 1965.

[19] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 302–309, New York, NY, USA, 1980. Assoc. Comput. Mach., New York.

[20] Leonid Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[21] Stephen R. Mahaney. Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS 1980)*, pages 54–60. IEEE Computer Soc., 1980.

[22] Pierre Marquis. Compile! In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press, 2015.

[23] Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT*, pages 125–129. IEEE Computer Soc., 1972.

[24] Juan Andrés Montoya and Moritz Müller. Parameterized random complexity. *Theory Comput. Syst.*, 52(2):221–270, 2013.

[25] Moritz Müller. *Parameterized Randomization*. PhD thesis, Albert Ludwigs Universität Freiburg, 2009.

[26] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2006.

[27] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[28] Andreas Pfandler, Stefan Rümmele, and Stefan Szeider. Backdoors to abduction. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*. AAAI Press/IJCAI, 2013.

[29] Steven David Prestwich. CNF encodings. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 75–97. IOS Press, 2009.

[30] Jörg Siekmann and Graham Wrightson, editors. *Automation of reasoning. Classical Papers on Computer Science 1967–1970*, volume 2. Springer Verlag, 1983.

[31] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[32] G. S. Tseitin. Complexity of a derivation in the propositional calculus. *Zap. Nauchn. Sem. Leningrad Otd. Mat. Inst. Akad. Nauk SSSR*, 8:23–41, 1968. English translation reprinted in [30].

[33] Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de