Bounded Fixed-Parameter Tractability and Reducibility

Rod Downey Jörg Flum Martin Grohe Mark Weyer

March 5, 2007

Abstract

We study a refined framework of parameterized complexity theory where the parameter dependence of fixed-parameter tractable algorithms is not arbitrary, but restricted by a function in some family $\mathscr{F}$. For every family $\mathscr{F}$ of functions, this yields a notion of *$\mathscr{F}$-fixed parameter tractability*. If $\mathscr{F}$ is the class of all polynomially bounded functions, then $\mathscr{F}$-fixed parameter tractability coincides with polynomial time decidability and if $\mathscr{F}$ is the class of all computable functions, $\mathscr{F}$-fixed parameter tractability coincides with the standard notion of fixed-parameter tractability. There are interesting choices of $\mathscr{F}$ between these two extremes, for example the class of all singly exponential functions.

In this article, we study the general theory of $\mathscr{F}$-fixed parameter tractability. We introduce a generic notion of reduction and two classes $\mathscr{F}$-W[P] and $\mathscr{F}$-XP, which may be viewed as analogues of NP and EXPTIME, respectively, in the world of $\mathscr{F}$-fixed parameter tractability.

1 Introduction

Parameterized complexity theory provides a framework for a refined complexity analysis of hard algorithmic problems. Instead of measuring the complexity solely as a function of the input length, as it is common in "classical" complexity theory, an additional parameter of the input is taken into account as well. The theory is mainly intended for addressing complexity issues in situations where the parameter is known to be small. In such situations, an algorithm whose running time is exponential, but only in the (small) parameter, may still be considered tractable. Formally, a parameterized problem is *fixed-parameter tractable* if it can be solved in time

$$f(k) \cdot n^{O(1)}, \tag{1.1}$$

where f is an arbitrary computable function. Here n denotes the input size and k the parameter. The class of all parameterized problems that are fixed-parameter tractable is denoted by FPT. For example, it is easy to see that the problem of deciding whether an n-vertex graph has a vertex cover of size k can be decided in time $O(2^k \cdot n)$. A *vertex cover* of a graph is a set of vertices such that every edge is incident with at least one vertex in this set. As a second, highly nontrivial, example due to Robertson and Seymour [13] we mention that the problem of deciding whether a k-vertex graph H is a minor of an n-vertex graph G can be decided in time $f(k) \cdot n^3$ for some computable function f. A graph H is a *minor* of a graph G if H can be obtained from a subgraph of G by contracting edges. No good bounds are known for the function f needed here; in Robertson and Seymour's proof it is a multiply iterated exponential. Nevertheless, both the vertex cover problem and the minor problem are fixed-parameter tractable if k is taken as the parameter. A generic example of a problem that is not fixed-parameter tractable is the problem of deciding whether a given deterministic Turing machine of size n halts in at most n^k steps. This problem is known to be complete for the parameterized complexity class XP of all parameterized problems that can be solved in time $O\big(n^{f(k)}\big)$, for some computable f. This class, which may be viewed as an analogue of the complexity class EXPTIME in the world of parameterized complexity theory, is known to contain FPT strictly. Intuitively, parameterized complexity is mainly concerned with the difference between FPT and XP, that is, between running times like $2^k \cdot n$ and n^k. Another typical problem that is believed not to be fixed-parameter tractable is the problem of deciding whether a given Boolean circuit of size n has a satisfying input assignment of Hamming weight k. Clearly this problem is in XP. It is complete for the parameterized complexity class W[P], which may be seen as an analogue of the complexity class NP in the world of parameterized complexity theory. Let us close this short introduction to parameterized complexity theory by pointing out an

aspect of the theory that may make it particularly attractive to computability theorists: The computability requirement and polynomial bound in an fpt running time as (1.1) interact in a nontrivial way. Some results in parameterized complexity require sophisticated arguments from computability theory, as can be seen in the last chapter of [8] and also later in this article. For more background in parameterized complexity theory, we refer the reader to the monographs [8, 9].

So in parameterized complexity theory a problem is viewed as "tractable" if it can be solved in time $f(k) \cdot n^{O(1)}$, for an arbitrary computable function f. However, even for small values of the parameter k, a running time such as

$$2^{2^{2^k}} \cdot n$$

is prohibitive. The graph minor problem mentioned above is an example of a fixed-parameter tractable problem for which the best known fixed-parameter tractable algorithm has a running time far worse than this. While for this problem there is still hope that better fixed-parameter tractable algorithms may exists, there are natural problems that are fixed-parameter tractable, but provably (modulo complexity theoretic assumptions) require a parameter dependence f that is nonelementary [11]. This led us to studying restrictions on the parameter dependence f [10]. For every class $\mathscr{F}$, there is a class $\mathscr{F}$-FPT of all problems that can be solved in time $f(k) \cdot n^{O(1)}$ for some $f \in \mathscr{F}$. Thus every class $\mathscr{F}$ of functions yields its own parameterized complexity theory. For classes $\mathscr{F}$ that do not comprise the class of all computable functions, we call these theories *bounded parameterized complexity theories*. The usual ("unbounded") class of all fixed-parameter tractable problems is $\mathscr{C}$-FPT for the class $\mathscr{C}$ of all computable functions. It is worth noting that the class PTIME is (essentially) $\mathscr{P}$-FPT for the class $\mathscr{P}$ of all polynomially bounded functions. Thus, in some sense, bounded parameterized complexity theories fill the gap between classical complexity theory and the usual unbounded parameterized complexity theory. The most natural restriction of the parameter dependence seems to be singly exponential, either by letting $\mathscr{F} = 2^{O(k)}$ or by letting $\mathscr{F} = 2^{\mathrm{poly}(k)}$; another interesting choice is $\mathscr{F} = 2^{o(k)}$. The corresponding theories have been studied in [6, 9, 10, 14]. This paper is not devoted to particular classes $\mathscr{F}$-FPT, but to a general theory.

At the heart of most branches of complexity theory are suitable notions of reducibility. We introduce generic reductions for the bounded parameterized complexity theories. The mathematical structure of the bounded theories starts to get interesting with these reductions. While for $\mathscr{F} \subseteq \mathscr{F}'$ we have $\mathscr{F}$-FPT $\subseteq \mathscr{F}'$-FPT, the corresponding reductions are incomparable. We study the reductions for a number of examples. For $2^{O(k)}$-FPT, they coincide with the reductions introduced in [10], and for $\mathscr{P}$-FPT, they coincide with polynomial time reductions. As an unpleasant surprise, we find that for the usual notion of fixed-parameter tractability, $\mathscr{C}$-FPT, our generic reductions do not coincide with the usual fpt reductions. Technically, this is our most difficult result, its proof requires a nontrivial priority argument (see Lemma 26).

We then introduce nondeterministic classes $\mathscr{F}$-W[P] that generalize the class W[P]. The intuition that W[P] is a parameterized analogue of the class NP is confirmed by our observation that $\mathscr{P}$-W[P] coincides with NP in the same way $\mathscr{P}$-FPT coincides with PTIME. We prove that a fundamental W[P]-completeness result due to Abrahamson et al. [1] generalizes to all $\mathscr{F}$-W[P], where $\mathscr{F}$ satisfies certain regularity and effectivity conditions. We also establish a general connection between the classes $\mathscr{F}$-W[P] and limited nondeterminism. Finally, we introduce classes $\mathscr{F}$-XP that generalize the class XP. Again, the intuition that XP is a parameterized analogue of EXPTIME is confirmed by the observation that $\mathscr{P}$-XP coincides with EXPTIME. For all regular $\mathscr{F}$, we prove that $\mathscr{F}$-FPT is strictly contained in $\mathscr{F}$-XP. We also prove a general completeness result for the classes $\mathscr{F}$-XP.

2 Preliminaries

$\mathbb{N}$ denotes the set of natural numbers (positive integers). For $m \leq n \in \mathbb{N}$ we let $[m,n] = \{m, m+1, \ldots, n\}$ and $[n] = [1,n]$. For tuples $\bar{m} = (m_1, \ldots, m_k), \bar{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$ we write $\bar{m} \leq \bar{n}$ if $m_i \leq n_i$ for all $i \in [k]$, and we write $\bar{m} < \bar{n}$ if $\bar{m} \leq \bar{n}$ and $\bar{m} \neq \bar{n}$.

2.1 Properties of functions

A function $f : \mathbb{N}^k \to \mathbb{N}$ is *nondecreasing* if $f(\bar{m}) \leq f(\bar{n})$ for all $\bar{m}, \bar{n} \in \mathbb{N}^k$ with $\bar{m} \leq \bar{n}$. The function f is *increasing* if for all $\bar{m}, \bar{n} \in \mathbb{N}^k$ with $\bar{m} < \bar{n}$ it holds that $f(\bar{m}) < f(\bar{n})$. The function f is *unbounded* if for

every $m \in \mathbb{N}$ there exists a tuple $\bar{n} \in \mathbb{N}^k$ such that $f(\bar{n}) \geq m$. For two functions $f, g : \mathbb{N}^k \to \mathbb{N}$ we write $f \leq g$ if for all $\bar{n} \in \mathbb{N}^k$ it holds that $f(\bar{n}) \leq g(\bar{n})$. We assume that the reader is familiar with the big-Oh and little-oh notation. Just to be clear, for a k-ary function $f : \mathbb{N}^k \to \mathbb{N}$, by $O(f)$ we denote the class of all functions $g : \mathbb{N}^k \to \mathbb{N}$ for which there exists a constant $c \in \mathbb{N}$ and an $n_0 \in \mathbb{N}$ such that $g(\bar{n}) \leq c \cdot f(\bar{n})$ for all $\bar{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$ with $\max\{n_1, \ldots, n_k\} \geq n_0$. Instead of $O(f)$, we also write $O(f(\bar{n}))$, and we shall also use notations like $g(O(f))$ or $g(O(f(\bar{n})))$. For example, $2^{O(n)}$ is the class of all functions $f : \mathbb{N} \to \mathbb{N}$ such that there is a constant $c \in \mathbb{N}$ and an $n_0 \in \mathbb{N}$ such that $f(n) \leq 2^{c \cdot n}$ for all $n \geq n_0$. We need an effective version of little-oh: For a computable function $f : \mathbb{N} \to \mathbb{N}$ we let $o^{\text{eff}}(f)$ be the class of all computable functions g for which there exists a computable function h such that for all $\ell \geq 1$ and all $n \geq h(\ell)$ we have $g(n) \leq f(n)/\ell$. It is easy to see that $g \in o^{\text{eff}}(f)$ if and only if there is a nondecreasing and unbounded computable function $h : \mathbb{N} \to \mathbb{N}$ and an $n_0 \in \mathbb{N}$ such that $g(n) \leq f(n)/h(n)$ for all $n \geq n_0$. We often denote the class $n^{O(1)}$ of all polynomially bounded functions by $\text{poly}(n)$ and also use notation like $2^{\text{poly}(n)}$.

A function $f : \mathbb{N}^2 \to \mathbb{N}$ is *bounded in terms of its first argument*, if there is a function $h : \mathbb{N} \to \mathbb{N}$, such that $f(k,n) \leq h(k)$ for all $k, n \in \mathbb{N}$. It is *computably bounded in terms of its first argument*, if h can be chosen to be computable.

Addition and multiplication of a functions are defined pointwise.

Definition 1. A class $\mathscr{F}$ of unary functions is *regular* if it is nonempty and satisfies the following conditions:

(1) For every $f \in \mathscr{F}$ there is a nondecreasing $g \in \mathscr{F}$ such that $f \leq g$.

(2) $\mathscr{F}$ is closed under addition and multiplication.

(3) $\mathscr{F}$ is closed under linear transformations of the arguments, that is, for every $f \in \mathscr{F}$ and $c \in \mathbb{N}$ the function g defined by $g(x) = f(c \cdot x)$ belongs to $\mathscr{F}$.

Example 2. The classes $O(1)$, $(\log n)^{O(1)}$, $\mathscr{P} := \text{poly}(n)$, $2^{o^{\text{eff}}(n)}$, $2^{o(n)}$, $2^{O(n)}$, $2^{\text{poly}(n)}$, the class $\mathscr{C}$ of all unary computable functions, and the class $\mathscr{A}$ of all unary functions are regular. The class $O(\log n)$ is not regular.

A function $f : \mathbb{N} \to \mathbb{N}$ is *time-constructible* if there is a deterministic Turing machine that, given n in unary, halts in exactly $f(n)$ steps. Observe that for every computable function $f : \mathbb{N} \to \mathbb{N}$ there is a time-constructible increasing function $g : \mathbb{N} \to \mathbb{N}$ such that $f \leq g$.

Definition 3. A class $\mathscr{F}$ of unary functions is *effectively regular* if it is regular and if for every $f \in \mathscr{F}$ there is a time-constructible nondecreasing function $g \in \mathscr{F}$ such that $f \leq g$.

2.2 Parameterized complexity

We view decision problems as languages over some finite alphabet, which we usually denote by Σ. A *parameterized problem* (over the alphabet Σ) is a pair (Q, κ), where $Q \subseteq \Sigma^*$ is a decision problem and $\kappa : \Sigma^* \to \mathbb{N}$, the *parameterization*, is a polynomial time computable function, where the result is encoded in unary. This implies that $\kappa(x)$ is polynomially bounded by $|x|$. A parameterized problem (Q, κ) is *fixed-parameter tractable* if there is an algorithm and a computable function f such that for every instance x the algorithm decides if $x \in Q$ in time

$$f(\kappa(x)) \cdot |x|^{O(1)}.$$

FPT denotes the class of all fixed-parameter tractable problems.

A *(many-one) fpt reduction* from a parameterized problem (Q, κ) over Σ to a parameterized problem (Q', κ') over Σ' is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that:

(R1) For all $x \in \Sigma^*$ we have $\big(x \in Q \iff R(x) \in Q'\big)$.

(R2) For all $x \in \Sigma^*$, the value $R(x)$ is computable in time

$$f(\kappa(x)) \cdot |x|^{O(1)}$$

for a computable $f : \mathbb{N} \to \mathbb{N}$.

(R3) There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

We write $(Q,\kappa) \leq^{\text{fpt}} (Q',\kappa')$ if there is an fpt reduction from (Q,κ) to (Q',κ'), and we write $(Q,\kappa) \equiv^{\text{fpt}} (Q',\kappa')$ if $(Q,\kappa) \leq^{\text{fpt}} (Q',\kappa')$ and $(Q',\kappa') \leq^{\text{fpt}} (Q,\kappa)$.

Besides FPT, we are interested in two further parameterized complexity classes: Let (Q,κ) be a parameterized problem over Σ. Then (Q,κ) belongs to the class XP if there is a computable function f and an algorithm that, given $x \in \Sigma^*$, decides if $x \in Q$ in time

$$O\left(|x|^{f(\kappa(x))}\right).$$

(Q,κ) belongs to the class W[P] if there are computable functions $f,g : \mathbb{N} \to \mathbb{N}$ and a nondeterministic Turing machine $\mathbb{M}$ deciding Q such that on every run with input $x \in \Sigma^*$ the machine $\mathbb{M}$ performs $f(\kappa(x)) \cdot |x|^{O(1)}$ steps, at most $g(\kappa(x)) \cdot \log|x|$ of them being nondeterministic. Observe that

$$\text{FPT} \subseteq \text{W[P]} \subseteq \text{XP}.$$

We close our short introduction into parameterized complexity theory with an example of a W[P]-complete problem. An input for a Boolean circuit C is an assignment of Boolean values to its input gates; the input is satisfying if the output of C on this input is 1. The circuit C is *k-satisfiable* if there is a satisfying assignment that sets exactly k input gates to '1'. The *parameterized weighted satisfiability problem for Boolean circuits* is defined as follows:

p-WSAT(CIRC)
Instance: A Boolean circuit C and a $k \in \mathbb{N}$.
Parameter: k.
Problem: Decide if C is k-satisfiable.

The problem p-WSAT(CIRC) is complete for W[P] under fpt-reductions [1]. As a matter of fact, W[P] has originally been defined as the downward closure of p-WSAT(CIRC) under fpt-reducibility.

For further background on parameterized complexity theory, we refer the reader to [8, 9].

3 The Bounded Framework

Definition 4. Let $\mathscr{F}$ be a class of unary functions. A parameterized problem (Q,κ) over the alphabet Σ is *$\mathscr{F}$-fixed-parameter tractable*, if there is an algorithm and a function $f \in \mathscr{F}$ such that for every instance $x \in \Sigma^*$ the algorithm decides if $x \in Q$ in time

$$f(\kappa(x)) \cdot |x|^{O(1)}.$$

$\mathscr{F}$-FPT denotes the class of all $\mathscr{F}$-fixed-parameter tractable problems.

Let $\mathscr{F}$ be a class of unary functions, Σ^* an alphabet and $\kappa : \Sigma^* \to \mathbb{N}$ a parameterization. We say that an algorithm with inputs x from Σ^* is $\mathscr{F}$-fpt (with respect to κ), if its running time is $f(\kappa(x)) \cdot |x|^{O(1)}$ for some $f \in \mathscr{F}$.

For technical reasons, we usually take the classes $\mathscr{F}$ defining $\mathscr{F}$-FPT to be regular.

Example 5. We obtain the standard notion of fixed-parameter tractability if we take as $\mathscr{F}$ the class $\mathscr{C}$ of all computable functions, that is,

$$\text{FPT} = \mathscr{C}\text{-FPT}.$$

Downey and Fellows [8] distinguish between our version of fixed-parameter tractability, which they call *strongly uniform fixed-parameter tractability*, and the weaker *uniform fixed-parameter tractability*. The class of all uniformly fixed-parameter tractable problems is $\mathscr{A}$-FPT for the class $\mathscr{A}$ of all unary functions.

In the following, we refer to the standard parameterized complexity theory as *unbounded parameterized complexity theory* and to the $\mathscr{F}$-theories for proper subclasses $\mathscr{F}$ of $\mathscr{C}$, as *bounded parameterized complexity theories*.

Example 6. Recall that $\mathscr{P} := \mathrm{poly}(k)$ denotes the class of all polynomially bounded unary functions. $\mathscr{P}$-FPT is essentially PTIME. More precisely, $\mathscr{P}$-FPT is the class of all parameterized problems (Q,κ), where $Q \in$ PTIME.

Observe that $\mathscr{P}$-FPT $= \mathscr{F}$-FPT for all nonempty classes $\mathscr{F} \subseteq \mathscr{P}$. In particular, $O(1)$-FPT $= \mathscr{P}$-FPT.

Example 7. The following notions of bounded fixed-parameter tractability have been studied in [9, 10, 14]:

$$2^{\mathrm{poly}(k)}\text{-FPT}, \qquad 2^{O(k)}\text{-FPT}, \quad \text{and} \quad 2^{o^{\mathrm{eff}}(k)}\text{-FPT}.$$

These classes are denoted by EXPT, EPT, and SUBEPT, respectively, in [9, 10].

Example 8. Recall that a function $f : \mathbb{N}^n \to \mathbb{N}$ is *elementary* if it can be formed from the successor function, addition, subtraction, and multiplication using composition, projections, bounded addition and bounded multiplication (of the form $\sum_{\ell \le m} g(n_1,\dots,n_k,\ell)$ and $\prod_{\ell \le m} g(n_1,\dots,n_k,\ell)$). Let $\mathscr{E}$ denote the class of all unary elementary functions. $\mathscr{E}$-FPT is another interesting notion of bounded fixed-parameter tractability.

It is well known that a function f is bounded by an elementary function if, and only if, it is bounded by a d-fold exponential function for some fixed d (see, for example, [7]). We use the following notation for iterated logarithms and exponentiations: For all $n \in \mathbb{N}$, we let $\log^{(0)} n := n$, $\exp^{(0)} n := n$, and for $d \ge 1$,

$$\log^{(d)} n := \log\log^{(d-1)} n, \qquad \exp^{(d)} n := 2^{\exp^{(d-1)} n}.$$

Then $\exp^{(O(1))}$ denotes the class of all functions f for which there exists a constant d and an n_0 such that $f(n) \le \exp^{(d)} n$ for all $n \ge n_0$. Thus

$$\mathscr{E}\text{-FPT} = \exp^{(O(1))}\text{-FPT}.$$

3.1 The classes $\mathscr{G}(\mathscr{F})$ of functions

Before we continue our theory with the introduction of suitable reductions, we need to introduce an important class of functions derived from $\mathscr{F}$.

Definition 9. Let $\mathscr{F}$ be a class of unary functions. Then $\mathscr{G}(\mathscr{F})$ is the class of all binary functions $g : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ such that for all $f \in \mathscr{F}$ there is a function $f' \in \mathscr{F}$ such that

$$f(g(k,n)) \le f'(k)\cdot n^{O(1)}$$

for all $k,n \in \mathbb{N}$.

The next lemma establishes a few basic properties of $\mathscr{G}(\mathscr{F})$ for regular $\mathscr{F}$.

Lemma 10. *Let $\mathscr{F}$ be a regular class of unary functions and $\mathscr{G} = \mathscr{G}(\mathscr{F})$.*

(1) For every $g \in \mathscr{G}$ there exists a nondecreasing $g' \in \mathscr{G}$ such that $g \le g'$.

(2) $\mathscr{G}$ is closed under addition.

(3) For all $g,g' \in \mathscr{G}$, all $f \in \mathscr{F}$, and all $c \in \mathbb{N}$, the function $h : \mathbb{N} \to \mathbb{N}$ defined by

$$h(k,n) = g(g'(k,n), f(k)\cdot n^c)$$

belongs to $\mathscr{G}$.

(4) There is a $g \in \mathscr{G}$ such that $g(k,n) \ge k$ for all $k,n \in \mathbb{N}$.

Proof. To prove (1), let $g \in \mathscr{G}$ and define $g' : \mathbb{N}^2 \to \mathbb{N}$ by $g'(k,n) = \max\{g(k',n') \mid k' \le k, n' \le n\}$. Then $g \le g'$, and g' is nondecreasing. We claim that $g' \in \mathscr{G}$. To see this, let $f \in \mathscr{F}$. Let $f' \in \mathscr{F}$ and $c \in \mathbb{N}$ such that $f(g(k,n)) \le f'(k)\cdot n^c$ for all $k,n \in \mathbb{N}$. By Definition 1(1) we may assume that f' is nondecreasing. Let $k' \le k$ and $n' \le n$ such that $g'(k,n) = g(k',n')$. Then

$$f(g'(k,n)) = f(g(k',n')) \le f'(k')\cdot(n')^c \le f'(k)\cdot n^c.$$

Hence $g' \in \mathscr{G}$, which completes the proof of (1).

To prove (2), let $g_1, g_2 \in \mathscr{G}$ and $f \in \mathscr{F}$. Let f' be defined by $f'(k) = f(2 \cdot k)$. Then $f' \in \mathscr{F}$ by Definition 1(3). We may assume that f and hence f' is nondecreasing. Furthermore, let $f_1, f_2 \in \mathscr{F}$ be such, that for all $k, n \in \mathbb{N}$ we have $f'(g_1(k,n)) \leq f_1(k) \cdot n^{O(1)}$ and $f'(g_2(k,n)) \leq f_2(k) \cdot n^{O(1)}$. Then

$$f((g_1 + g_2)(k,n)) \leq \max\{f'(g_1(k,n)), f'(g_2(k,n))\} \leq (f_1 + f_2)(k) \cdot n^{O(1)}$$

By Definition 1(2), this shows, that $g_1 + g_2 \in \mathscr{G}$.

To prove (3), let $g, g' \in \mathscr{G}$ and $f \in \mathscr{F}$. Let $f' \in \mathscr{F}$ be given and let $f'', f''' \in \mathscr{F}$ be such that for all $k, n \in \mathbb{N}$ it holds that $f'(g(k,n)) \leq f''(k) \cdot n^{O(1)}$ and $f''(g'(k,n)) \leq f'''(k) \cdot n^{O(1)}$. Then

$$f'(h(k,n)) \leq f''(g'(k,n)) \cdot (f(k) \cdot n^{O(1)})^{O(1)} \leq f'''(k) \cdot f(k)^{O(1)} \cdot n^{O(1)}.$$

By Definition 1(2), this shows that $h \in \mathscr{G}$.

(4) follows immediately from the definition of $\mathscr{G}$. □

3.2 Reductions

Let us fix a regular class $\mathscr{F}$ of unary functions. As in the unbounded theory, to compare the complexities of parameterized problems that are not $\mathscr{F}$-fixed-parameter tractable, we need a notion of (many-one) reduction.[1]

Let (Q, κ) and (Q', κ') be parameterized problems over the alphabets Σ and Σ', respectively. Clearly, a (many-one) reduction from (Q, κ) to (Q', κ') is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that:

(1) For all $x \in \Sigma^*$ we have $(x \in Q \iff R(x) \in Q')$.

The crucial property we expect R to have is:

If $\mathbb{A}$ is an $\mathscr{F}$-fpt algorithm solving (Q', κ'), then the algorithm $\mathbb{B}$ that on input $x \in \Sigma^*$

$$\text{first computes } R(x) \text{ and then decides if } R(x) \in Q' \text{ with } \mathbb{A} \tag{3.1}$$

is an $\mathscr{F}$-fpt algorithm, too.

In particular, this implies that

(2) R is computable by an $\mathscr{F}$-fpt algorithm.

So, assume that $R(x)$ is computable in time $f(\kappa(x)) \cdot |x|^{O(1)}$ and $\mathbb{A}$ solves $x' \in Q'$ in time $f'(\kappa'(x')) \cdot |x'|^{O(1)}$ for some $f, f' \in \mathscr{F}$. Then according to (3.1) the decision algorithm $\mathbb{B}$ has running time

$$\begin{aligned} f(\kappa(x)) \cdot |x|^{O(1)} + f'(\kappa'(R(x))) \cdot |R(x)|^{O(1)} & \\ \leq f(\kappa(x)) \cdot |x|^{O(1)} + f'(\kappa'(R(x))) \cdot f(\kappa(x))^{O(1)} \cdot |x|^{O(1)}. & \end{aligned} \tag{3.2}$$

Since $\mathscr{F}$ is regular, the running time (3.2) can be bounded by $h(\kappa(x)) \cdot |x|^{O(1)}$ for some $h \in \mathscr{F}$ if:

(3) $\kappa'(R(x)) \leq g(\kappa(x), x)$ for some $g \in \mathscr{G}(\mathscr{F})$.

Altogether, we define:

Definition 11. Let $\mathscr{F}$ be a regular class of functions. Let (Q, κ) and (Q', κ') be parameterized problems over the alphabets Σ and Σ', respectively. A mapping $R : \Sigma^* \to (\Sigma')^*$ is an *$\mathscr{F}$-fpt reduction* from (Q, κ) to (Q', κ') if

(1) For all $x \in \Sigma^*$ we have $(x \in Q \iff R(x) \in Q')$.

[1] It is straightforward to define a "Turing version" of all notions of (many-one) reduction considered in this paper, but this would not convey any new insights into the issues studied here.

(2) R is computable by an $\mathscr{F}$-fpt algorithm.

(3) There is a $g \in \mathscr{G}(\mathscr{F})$ such that $\kappa'(R(x)) \leq g(\kappa(x), |x|)$ for all $x \in \Sigma^*$.

We write $(Q,\kappa) \leq^{\mathscr{F}} (Q',\kappa')$ if there is an $\mathscr{F}$-fpt reduction from (Q,κ) to (Q',κ').

The remarks preceding this definition show:

Proposition 12. *If* $(Q,\kappa) \leq^{\mathscr{F}} (Q',\kappa')$ *and* $(Q',\kappa') \in \mathscr{F}$-FPT*, then* $(Q,\kappa) \in \mathscr{F}$-FPT.

To study and construct $\mathscr{F}$-fpt reductions it is crucial to determine the class $\mathscr{G}(\mathscr{F})$. Even for simple $\mathscr{F}$, the class of $\mathscr{G}(\mathscr{F})$ can be surprisingly complicated. In the following, we study $\mathscr{G}(\mathscr{F})$ for the examples considered earlier.

Example 13. $\mathscr{G}(O(1))$ is the class of all binary functions. Thus an $O(1)$-fpt reduction from (Q,κ) to (Q',κ') is just a polynomial time reduction from Q to Q'. To see this, note that condition (3) in Definition 11 can be satisfied by letting

$$g(k,n) = \max\{\kappa'(R(x)) \mid |x| \leq n\}.$$

Example 14. $\mathscr{G}(\mathscr{P})$ is the class of all binary functions, which are polynomially bounded in terms of their arguments. While this is different from $\mathscr{G}(O(1))$, $\mathscr{P}$-fpt reductions coincide with $O(1)$-fpt reductions.

With the single technical exception of $\mathscr{G}(\cdot)$, throughout this paper it will always be the case, that $O(1)$ and $\mathscr{P}$ lead to the same concepts. Also recall Example 6 in this light.

Example 15. $\mathscr{G}(2^{O(k)})$ is the class of all functions $g : \mathbb{N}^2 \to \mathbb{N}$ such that $g(k,n) \leq c \cdot (k + \log n)$ for some constant c, or more concisely,

$$\mathscr{G}(2^{O(k)}) = O(k + \log n).$$

To see the inclusion from left to right, let $g \in \mathscr{G}(2^{O(k)})$. Then there exist constants $d_1, d_2 > 0$ such that for all sufficiently large n,k it holds that

$$2^{g(k,n)} \leq 2^{d_1 \cdot k} \cdot n^{d_2}.$$

Taking logarithms, we obtain $g(k,n) \leq (d_1 + d_2) \cdot (k + \log n)$.

For the converse, let $g \in O(k+\log n)$ and $f \in 2^{O(k)}$ be given, say $g(k,n) \leq c \cdot (k+\log n)$ and $f(k) \leq 2^{d \cdot k}$. Then,

$$f(g(k,n)) \leq 2^{d \cdot c \cdot (k + \log n)} = 2^{d \cdot c \cdot k} \cdot n^{d \cdot c} = 2^{O(k)} \cdot n^{O(1)}.$$

Thus $2^{O(k)}$-ftp reductions are precisely the ept reductions introduced in [10].

Before we continue with more examples, let us point out that in general for classes $\mathscr{F} \subset \mathscr{F}'$ of functions, $\mathscr{F}$-ftp reductions and $\mathscr{F}'$-fpt reductions are incomparable. That is, neither the implication $(Q,\kappa) \leq^{\mathscr{F}} (Q',\kappa') \implies (Q,\kappa) \leq^{\mathscr{F}'} (Q',\kappa')$ nor the reverse implication $(Q,\kappa) \leq^{\mathscr{F}'} (Q',\kappa') \implies (Q,\kappa) \leq^{\mathscr{F}} (Q',\kappa')$ holds for all parameterized problems (Q,κ) and (Q',κ'). For example, it is observed in [9, 10] that $\leq^{O(1)}$, $\leq^{2^{O(k)}}$, and $\leq^{\mathscr{C}}$ are incomparable.

Example 16. Maybe surprisingly,

$$\mathscr{G}(2^{o(k)}) = \mathscr{G}(2^{O(k)}) = O(k + \log n). \tag{3.3}$$

To see that $O(k+\log n) \subseteq \mathscr{G}(2^{o(k)})$, note that $2^{o(k+\log n)} \subseteq 2^{o(k)+O(\log n)} = 2^{o(k)} \cdot n^{O(1)}$. To see that $\mathscr{G}(2^{o(k)}) \subseteq O(k+\log n)$, let $g \in \mathscr{G}(2^{o(k)})$ and suppose for contradiction that for all $c \in \mathbb{N}$ there are $k(c), n(c) \in \mathbb{N}$ such that $g(k(c),n(c)) > c \cdot (k(c) + \log(n(c)))$. Let $h : \mathbb{N} \to \mathbb{N}$ be defined by $h(i) = g(k(i),n(i))$. Without loss of generality we may assume that h is increasing. Let $f : \mathbb{N} \to \mathbb{N}$ be defined by

$$\begin{cases} f(j) = 1 & \text{for all } j \leq h(1), \\ f(j) = i & \text{for all } i > 1, h((i-1)^2) < j \leq h(i^2). \end{cases}$$

Then f is a nondecreasing unbounded function, and for all $i \in \mathbb{N}$,

$$f(h(i^2)) = i.$$

By the definition of $\mathscr{G}(2^{o(k)})$, there exists a nondecreasing unbounded $f' : \mathbb{N} \to \mathbb{N}$ and a $c \in \mathbb{N}$ such that

$$2^{g(k,n)/f(g(k,n))} \leq 2^{k/f'(k)} \cdot n^c \qquad \text{for all } k,n \in \mathbb{N}.$$

Taking logarithms, we get

$$\frac{g(k,n)}{f(g(k,n))} \leq \frac{k}{f'(k)} + c \cdot \log n \leq c \cdot (k + \log n).$$

For $k = k(c^2), n = n(c^2)$, this leads to a contradiction:

$$\frac{g(k(c^2),n(c^2))}{f(g(k(c^2),n(c^2)))} = \frac{h(c^2)}{f(h(c^2))} = \frac{h(c^2)}{c} > \frac{c^2(k+\log n)}{c} = c \cdot (k + \log n).$$

This completes the proof of (3.3).

It follows that for all parameterized problems (Q,κ) and (Q',κ') we have

$$(Q,\kappa) \leq^{2^{o(k)}} (Q',\kappa') \implies (Q,\kappa) \leq^{2^{O(k)}} (Q',\kappa').$$

Lemma 17. *Let $\mathscr{F}$, Σ, Σ', (Q,κ), and (Q',κ') be as in Definition 11. Then a mapping $R : \Sigma^* \to (\Sigma')^*$ is an $\mathscr{F}$-fpt reduction from (Q,κ) to (Q',κ') if and only if it satisfies conditions (1) and (2) of Definition 11 and the following, more restrictive, version of condition (3):*

(3') There is a nondecreasing computable $g \in \mathscr{G}(\mathscr{F})$ such that $\kappa'(R(x)) \leq g(\kappa(x),|x|)$ for all $x \in \Sigma^*$.

Proof. We define $g : \mathbb{N}^2 \to \mathbb{N}$ by

$$g(k,n) = \max\{\kappa'(R(x)) \mid x \in \Sigma^* \text{ with } |x| \leq n \text{ and } \kappa(x) \leq k\}.$$

Clearly, g is nondecreasing and computable. It is in $\mathscr{G}(\mathscr{F})$, because for a regular $\mathscr{F}$, for every $g_1 \in \mathscr{G}(\mathscr{F})$ there is a nondecreasing $g_2 \in \mathscr{G}(\mathscr{F})$ such that $g_1 \leq g_2$, and furthermore, $\mathscr{G}(\mathscr{F})$ is downward closed with respect to $\leq$. □

Example 18. It is not hard to see that the class $\mathscr{G}(2^{o^{\text{eff}}(k)})$ contains functions that are not in $\mathscr{G}(2^{o(k)})$. The interesting fact, however, is that this does not affect the corresponding notion of reduction. That is, for all parameterized problems (Q,κ) and (Q',κ') we have

$$(Q,\kappa) \leq^{2^{o^{\text{eff}}(k)}} (Q',\kappa') \implies (Q,\kappa) \leq^{2^{o(k)}} (Q',\kappa'). \tag{3.4}$$

To see this, recall the proof that $\mathscr{G}(2^{o(k)}) \subseteq O(k+\log n)$ from Example 16. Observe that if $g \in \mathscr{G}(2^{o^{\text{eff}}(k)})$ is nondecreasing and computable with $g \notin O(k+\log n)$, then the functions $k,n : \mathbb{N} \to \mathbb{N}$ with $g(k(c),n(c)) > c \cdot (k+\log n)$ can be chosen to be computable. Then the functions h, f defined in the proof of Example 16 are computable, and therefore

$$2^{g(k,n)/f(g(k,n))} \in 2^{o^{\text{eff}}(k)} \cdot n^c.$$

This leads to a contradiction. Therefore, all nondecreasing and computable $g \in \mathscr{G}(2^{o^{\text{eff}}(k)})$ are contained in $O(k+\log n)$. Now (3.4) follows from Lemma 17.

Example 19. We have

$$\mathscr{G}(2^{\text{poly}(k)}) = \bigcap_{d \in \mathbb{N}} (\text{poly}(k) \cdot (\log n)^{\frac{1}{d}}). \tag{3.5}$$

To understand this notation, remember that $\text{poly}(k) \cdot (\log n)^{\frac{1}{d}}$ denotes a class of functions. Hence $\mathscr{G}(2^{\text{poly}(k)})$ is the class of all functions $g : \mathbb{N}^2 \to \mathbb{N}$ with the following property: For all $d \in \mathbb{N}$ there is some polynomial $p_d(X)$ such that for all $k,n \in \mathbb{N}$ we have $g(k,n) \leq p_d(k) \cdot (\log n)^{\frac{1}{d}}$.

Let us prove (3.5): For the inclusion from left to right, let $g \in \mathscr{G}(2^{\text{poly}(k)})$ and $d \in \mathbb{N}$ be given. Then, for suitable constants $c_1, c_2 \in \mathbb{N}$ and all $k,n \in \mathbb{N}$, we have $2^{(g(k,n))^d} \leq 2^{k^{c_1}} \cdot n^{c_2}$. Takings logarithms and $\frac{1}{d}$th powers, we even obtain $g(k,n) \leq k^{\frac{c_1}{d}} + (c_2 \cdot \log n)^{\frac{1}{d}}$.

For the converse, let g be in the right side set and let $f \in 2^{\mathrm{poly}(k)}$, say $f(k) \leq 2^{d+k^d}$. We need to bound $f(g(k,n))$. We consider the bound $g(k,n) \leq p_{2d}(k) \cdot (\log n)^{\frac{1}{2d}}$. As $a \cdot b \leq \max(|a|,|b|)^2 \leq a^2 + b^2$ for all nonnegative a, b, we obtain

$$f(g(k,n) \leq 2^{d+(p_{2d}(k)\cdot(\log n)^{\frac{1}{2d}})^d} \leq 2^{d+p_{2d}(k)^{2d}+\log n} = 2^{\mathrm{poly}(k)} \cdot n.$$

This completes the proof of (3.5).

Let us remark that the class $\mathrm{poly}(k) \cdot (\log n)^{o(1)}$, which might have been a first guess for $\mathscr{G}(2^{\mathrm{poly}(k)})$, is strictly contained in $\mathscr{G}(2^{\mathrm{poly}(k)})$. It is not hard to prove that the function $g : \mathbb{N}^2 \to \mathbb{N}$ defined by $g(k,n) = \min_{d\in\mathbb{N}} \lceil k^d + (\log n)^{\frac{1}{d}} \rceil$ is in $\mathscr{G}(2^{\mathrm{poly}(k)}) \setminus \big(\mathrm{poly}(k) \cdot (\log n)^{o(1)}\big)$.

Example 20. $\mathscr{G}(\mathscr{E}) = \bigcap_{d\in\mathbb{N}} ((\exp^{(O(1))} k) \cdot \log^{(d)} n)$. We leave the straightforward proof to the reader.

Example 21. For the class $\mathscr{A}$ of all unary functions, $\mathscr{G}(\mathscr{A})$ is the class of all functions $g : \mathbb{N}^2 \to \mathbb{N}$ which are bounded in terms of their first argument (recall that this means that $g(k,n) \leq h(k)$ for some function h and all $k, m\mathbb{N}$). Clearly, all binary functions g bounded in their first argument are contained in $\mathscr{G}(\mathscr{A})$. To see the converse, suppose for contradiction, that $g \in \mathscr{G}(\mathscr{A})$ and that for $k_0 \in \mathbb{N}$ the function $h_1 : \mathbb{N} \to \mathbb{N}$ defined by $h_1(n) = g(k_0, n)$ is unbounded. Let $h_2 : \mathbb{N} \to \mathbb{N}$ be nondecreasing, such that for all $m \in \mathbb{N}$ we have $h_1(h_2(m)) \geq m$. Now define $f : \mathbb{N} \to \mathbb{N}$ by $f(m) = 2^{h_2(m)}$. By Definition 9, the term $f(g(k_0,n))$ is polynomially bounded by n. But for n of the form $n = h_2(m)$ we have

$$f(g(k_0,n)) = f(h_1(h_2(m))) \geq f(m) = 2^n.$$

Definition 22. For a nondecreasing unbounded function $f : \mathbb{N} \to \mathbb{N}$, we define its "inverse" $\iota_f : \mathbb{N} \to \mathbb{N}$ by

$$\iota_f(n) = \max\big(\{1\} \cup \{m \mid f(m) \leq n\}\big). \tag{3.6}$$

Then for all $n \in \mathbb{N}$ we have $\iota_f(f(n)) \geq n$ and $f(\iota_f(n)) \leq \max\{f(1), n\}$. Furthermore, ι_f is unbounded, and if f is computable then so is ι_f.

Example 23. For the class $\mathscr{C}$ of all computable unary functions, $\mathscr{G}(\mathscr{C})$ contains functions g that are not bounded in terms of their first argument.

To see this, let $(f_i)_{i\in\mathbb{N}}$ be an enumeration of $\mathscr{C}$. Define $h : \mathbb{N} \to \mathbb{N}$ by $h(m) = \max\{f_i(m) \mid i \leq m\}$, and $g : \mathbb{N}^2 \to \mathbb{N}$ by $g(k,n) = \iota_h(n)$. Clearly, g is not bounded in terms of its first argument. To show $g \in \mathscr{G}(\mathscr{C})$, let $f \in \mathscr{C}$ be arbitrary, say, $f = f_i$. Then $h(m) \geq f(m)$ for all $m \geq i$. Hence, for sufficiently large n and all k we have

$$f(g(k,n)) \leq h(\iota_h(n)) \leq n + h(1).$$

Then, of course, there is a suitable constant function f', such that $f(g(k,n)) \leq f'(k) \cdot n$ for all k, n.

On the other hand, the next lemma states that the example depends on the non-computability of g. One might further hope that all computable functions in $\mathscr{G}(\mathscr{C})$ are even computably bounded in terms of their first argument. By Lemma 17, this would prove that $\mathscr{C}$-fpt reducibility coincides with fpt reducibility. However, in the following subsection we shall see that this is not that case, that is, there are computable functions in $\mathscr{G}(\mathscr{C})$ that are not computably bounded in terms of their first argument, and $\mathscr{C}$-fpt reducibility does not coincide with fpt reducibility. In this light it is not immediate, that every $\mathscr{C}$-fpt reduction also is an $\mathscr{A}$-fpt reduction. However, this follows from the next lemma.

Lemma 24. *Every computable function in $\mathscr{G}(\mathscr{C})$ is bounded in terms of its first argument.*

Proof. For contradiction, let $g \in \mathscr{G}(\mathscr{C})$ be computable and let $k \in \mathbb{N}$ be such, that the function $g_k : \mathbb{N} \to \mathbb{N}$ defined by $g_k(n) = g(k,n)$ is unbounded. Without loss of generality, g is nondecreasing. Then let $f \in \mathscr{C}$ be unbounded and nondecreasing, such that $\iota_f = g_k^{o(1)}$. For example, we could set $f(m) = \min\{n \in \mathbb{N} \mid m \leq g_k(\log n)\}$.

Now, let $f' \in \mathscr{C}$ be such, that $f(g(k,n)) \leq f'(k) \cdot n^{O(1)}$. Then for sufficiently large n we have

$$g_k(n) = g(k,n) \leq \iota_f(f(g(k,n))) \leq \iota_f(f'(k) \cdot n^{O(1)}) < g_k(n),$$

yielding the contradiction. □

This lemma together with Lemma 17 and Example 21 imply:

Corollary 25. *For all parameterized problems (Q,κ) and (Q',κ'):*

$$(Q,\kappa) \leq^{\mathscr{C}} (Q',\kappa') \implies (Q,\kappa) \leq^{\mathscr{A}} (Q',\kappa').$$

3.3 $\mathscr{C}$-fpt reductions vs fpt reductions

Lemma 26. *There is a computable function $g \in \mathscr{G}(\mathscr{C})$ that is not computably bounded in terms of its first argument, that is, for all computable functions $f \in \mathscr{C}$ there are $k,n \in \mathbb{N}$ with*

$$g(k,n) > f(k).$$

Proof. We fix some effective enumeration of the computable unary partial functions. Let both d_i and a_i denote the ith function in this enumeration. Let $T = \{i \mid a_i \text{ total}\}$ be the set of indices of the total computable functions.

Requirements: Our goal is to construct a computable function $g : \mathbb{N}^2 \to \mathbb{N}$ and, for all $i \in \mathbb{N}$, numbers $k_i, n_i \in \mathbb{N}$ and computable total functions $h_i : \mathbb{N} \to \mathbb{N}$, such that

I. $g(k_i,n_i) > d_i(k_i)$ for all $i \in T$, and

II. for all $j \in T$ there is some $c \in \mathbb{N}$, such that $a_j(g(k,n)) \leq \max(h_j(k),n,c)$ for all $k,n \in \mathbb{N}$.

We describe an infinite computation which will construct all necessary witnesses. This means that the computation is only guaranteed to provide k_i and n_i if $i \in T$ and h_j if $j \in T$.

Along with k_i and n_i, the computation will also construct a number v_i, which will be a lower bound on $g(k_i,n_i)$. Indeed, we define g by

$$g(k,n) = \max(\{1\} \cup \{v_i \mid k_i,n_i \text{ are defined and } k = k_i, n = n_i\}). \tag{3.7}$$

The computation will be carried out in stages, and each stage will require only finite computation time. We will make sure, that k_i,n_i, if defined at all, are defined not later than at stage n_i. This implies that g is computable, because to compute $g(k,n)$ we only have to simulate the first n stages of the construction.

The computation has to make some assumptions about totality of the a_j. That is, at stage s it will "guess" how $T \cap \{1,\dots i\}$ looks for $i \leq s$ and act accordingly. Alternatively, we may think of the computation as being organized as an infinite binary tree, where different branches of the tree correspond to different assumptions. We denote the branches, or guesses for $T \cap \{1,\dots i\}$, by B. To find appropriate values for k_i, v_i, and n_i, the computation will repeatedly introduce candidates $k_{i,B}$ and $v_{i,B}$ for branches $B \subseteq \{1,\dots,i\}$.

The last pieces of data constructed by the computation are numbers $\ell_j \in \mathbb{N}$ for $j \in \mathbb{N}$. The number ℓ_j is a convergence requirement for a_j: Only if a_j is witnessed to be total at least on $\{1,\dots,\ell_j\}$, something happens for a_j. One particular thing that happens is that ℓ_j is increased. Accordingly, each non-total a_j will eventually be recognized as such and no longer disturb the computation.

To summarize: The computation will construct numbers

$$k_i, v_i, n_i, k_{i,B}, v_{i,B}, \ell_i,$$

for $i \in \mathbb{N}$ and $B \subseteq \{1,\dots,i\}$, and functions

$$h_j$$

that will be total for all $j \in T$. As said before, the computation will be carried out in stages. For a number $x \in \{k_i, v_i, n_i, k_{i,B}, v_{i,B}, \ell_i\}$ we write $x \searrow s$ to denote that x is defined at the beginning of stage s. Occasionally, we also use $x \searrow s$ to denote the value of x at beginning of stage s. For a computable function f and an argument x, we write $f(x) \searrow s$ to denote that $x < s$, $f(x) < s$, and the computation of f on input x halts in at most s steps. We use $x \nearrow s$ and $f(x) \nearrow s$ to denote that $x \searrow s$ respectively $f(x) \searrow s$ does not hold.

A function a_j is *active* at stage s of the construction if $\ell_j \searrow s$ and $a_j(m) \searrow s$ for all $m \leq (\ell_j \searrow s)$.

Construction: At stage s, the computation proceeds as follows:

(1) For all $j \leq s$ such that $\ell_j \nearrow s$, set $\ell_j \leftarrow 1$.

(2) For all $j \leq s$ such that a_j is active, set $\ell_j \leftarrow s$.

(3) For all $j \leq s$ such that a_j is active and all $k \leq \ell_j$ such that $h_j(k) \nearrow s$, set $h_j(k) \leftarrow s$.

(4) For all $i \leq s$ and all $B \subseteq \{1,\ldots,i\}$ such that $k_{i,B} \nearrow s$, set $k_{i,B} \leftarrow s$.

(5) For all $i \leq s$ and all $B \subseteq \{1,\ldots,i\}$ such that $k_{i,B} \searrow s$, $v_{i,B} \nearrow s$, and $d_i(k_{i,B}) \searrow s$, set $v_{i,B} \leftarrow s$.

(6) For all $j \leq s$ such that a_j is active, all i with $j \leq i \leq s$, and all $B \subseteq \{1,\ldots,i\}$ such that $j \notin B$, delete $k_{i,B}$ and $v_{i,B}$ (if defined).

(7) For all $i \leq s$, if there is a $B \subseteq \{1,\ldots,i\}$ such that $v_{i,B} \searrow s$, $n_i \nearrow s$, and $a_j(v_{i,B}) \searrow s$ for all $j \in B$, then take the lexicographically first such B and set $k_i \leftarrow k_{i,B}$, $v_i \leftarrow v_{i,B}$, and $n_i \leftarrow s$.

Correctness: Observe first that at each stage of the construction the computation is finite and that the numbers k_i, v_i, n_i, and $h_i(k)$ for $i,k \in \mathbb{N}$, are defined at most once during the construction. Only the $k_{i,B}$, $v_{i,B}$, and ℓ_i can be deleted and redefined, possibly infinitely often.

Let us now prove that requirement I is satisfied. Let $i \in T$. We claim that eventually k_i, v_i, and n_i are defined. Let $B := T \cap \{1,\ldots,i\}$ be the correct "totality guess". Then $k_{i,B}$ and $v_{i,B}$ can only be deleted finitely many times in step 6. As d_i is total, after every deletion, $v_{i,B}$ will be redefined in step 5 at some later stage. As a_j is total for all $j \in B$, the preconditions of step 7 will eventually be fulfilled, if not earlier for some B', then for B. Now let B' be the guess which leads to the final values of k_i, v_i and n_i. Let s be the stage where $v_{i,B}$ obtains the value, which later is taken for v_i. By the definition of g (see (3.7)), we have

$$g(k_i,n_i) \geq v_i = (v_{i,B} \searrow (s+1)) = s > d_i(k_{i,B} \searrow s) = d_i(k_i),$$

as required.

To prove that requirement II is satisfied, let a_j be total and k,n be given. If $g(k,n) = 1$, then it is sufficient to have $c \geq a_j(1)$. Otherwise there is some $i \in \mathbb{N}$, such that $k = k_i$, $n = n_i$, and $g(k,n) = v_i$. We shall prove that for all i we have

$$a_j(v_i) \leq \max\{h_j(k_i), n_i, c\}. \tag{3.8}$$

Letting $c \geq \max\{a_j(v_i) \mid i < j\}$, we take care of the finitely many $i < j$. In the following, we assume that $j \leq i$. Let B be the totality assumption which led to the definition of k_i, v_i, n_i in step 7, and let s be the stage where this happened. If $j \in B$, then the precondition of 7 implies

$$a_j(v_i) \leq s = n_i.$$

Otherwise, let s_1 be minimal such that $s_1 \geq k_i$ and such that a_j is active at stage s_1. Then $s < s_1$, because $k_{i,B}$ will be deleted in step 6 at stage s_1. Hence $v_i \leq s < s_1$. In step 2 of stage s_1, ℓ_j will be set to s_1; this is the first time that $\ell_j \geq k_i$. Thus the stage s_2 where $h_j(k_i)$ will be defined comes after s_1. As a_j is active at stage s_2 and $v_i < s_1 = (\ell_j \searrow s_2)$, we have

$$a_j(v_i) < s_2 = h_j(k_i).$$

This proves (3.8). □

The next lemma shows that a function g as constructed in the previous lemma can even be chosen to be polynomial time computable. In the following, it will be convenient (but inessential) to encode all integers in unary.

Lemma 27. *There is a polynomial time computable nondecreasing function $g \in \mathscr{G}(\mathscr{C})$ such that for all computable functions $f \in \mathscr{C}$ there are $k,n \in \mathbb{N}$ with*

$$g(k,n) > f(k).$$

Proof. Let g' be as in Lemma 26. Without loss of generality, we may assume that g' is nondecreasing. Let $f : \mathbb{N} \to \mathbb{N}$ be increasing and time-constructible, such that $g'(k,n)$ can be computed in time at most $f(\max(k,n))$. Then ι_f, as defined in (3.6), is computable in quadratic time. Now define g by $g(k,n) = g'(\iota_f(k), \iota_f(n))$. As a composition of computable functions, g is computable itself. Moreover, the time needed to compute g on input (k,n), is the quadratic time, quadratic in the arguments, to compute $\iota_f(k)$ and $\iota_f(n)$, plus a time bounded by

$$f(\max(\iota_f(k), \iota_f(n))) = f(\iota_f(\max(k,n))) \leq \max(k,n,f(1)).$$

As $\iota_f(n) \leq n$ for all n, we have $g \leq g'$ and hence $g \in \mathscr{G}(\mathscr{C})$. It remains to show that $g(k,n)$ is not computably bounded in terms of k. Assume otherwise that there is a computable h such that $g(k,n) \leq h(k)$ for all k,n. But then

$$g'(k,n) \leq g'(\iota_f(f(k)), \iota_f(f(n))) = g(f(k), f(n)) \leq h(f(k)).$$

Hence g' would be bounded in terms of its first argument by the computable function $h \circ f$, which is a contradiction. □

Theorem 28. *There are parameterized problems (Q,κ) and (Q',κ') in* XP *such that (Q,κ) is $\mathscr{C}$-fpt reducible to (Q',κ'), but not fpt reducible.*

Proof. Choose g according to Lemma 27. Without loss of generality we may assume that g is nondecreasing and $g(k,n) \geq k$ for all $k \in \mathbb{N}$; simply replace $g(k,n)$ by $\max(\{k\} \cup \{g(k',n') \mid k' \leq k, n' \leq n\})$ if not. Let (Q,κ) and (Q',κ') be the following parameterized problems:

Input:	A Turing machine $\mathbb{M}$ and $k \in \mathbb{N}$.
Parameter:	k
Problem:	Decide whether $\mathbb{M}$ halts on empty input in at most $\lvert\mathbb{M}\rvert^k$ steps.

Input:	A Turing machine $\mathbb{M}$ and $k \in \mathbb{N}$.
Parameter:	$g(k, \lvert\mathbb{M}\rvert)$
Problem:	Decide whether $\mathbb{M}$ halts on empty input in at most $\lvert\mathbb{M}\rvert^k$ steps.

Note that actually we have $Q = Q'$; the two problems only differ in their parameterization.

Clearly, the identity function is a $\mathscr{C}$-fpt reduction from (Q,κ) to (Q',κ'). Just simulating $\lvert\mathbb{M}\rvert^k$ steps of $\mathbb{M}$ suffices to have $(Q,\kappa) \in$ XP, and, as $g(k,n) \geq k$, also to have $(Q',\kappa') \in$ XP.

Let Σ be the alphabet of Q. Suppose for contradiction that $R : \Sigma^* \to \Sigma^*$ is an fpt reduction from (Q,κ) to (Q',κ'). Let $f,h : \mathbb{N} \to \mathbb{N}$ and $c \in \mathbb{N}$ be such that $R(x)$ is computable in time $f(\kappa(x)) \cdot |x|^c$ and that $\kappa'(R(x)) \leq h(\kappa(x))$ for all $x \in \Sigma^*$. For a Turing machine $\mathbb{M}$ and $k \in \mathbb{N}$, let

$$\big(\mathbb{M}'(\mathbb{M},k), k'(\mathbb{M},k)\big) = R((\mathbb{M}, c \cdot k + 1)).$$

We now distinguish between two cases, both leading to a contradiction:

Case 1: There are $k,n \in \mathbb{N}$ such that for all Turing machines $\mathbb{M}$ it holds that $|\mathbb{M}'(\mathbb{M},k)| \leq n$ or $k'(\mathbb{M},k) \leq k$.

Choose such k,n. We claim that there is an algorithm that decides in time

$$O(m^{c \cdot k})$$

whether a given Turing machine $\mathbb{M}$ of size m halts on empty input in at most $m^{c \cdot k+1}$ steps. This contradicts the time hierarchy theorem.

For convenience we use the shorthands $\mathbb{M}' = \mathbb{M}'(\mathbb{M},k)$ and $k' = k'(\mathbb{M}',k)$. The algorithm first computes $\mathbb{M}'$ and k' in time $O(m^c)$. If $|\mathbb{M}'| \leq n$, then it simply uses a lookup table to decide whether $\mathbb{M}'$ halts in at most $|\mathbb{M}'|^{k'}$ steps. Note that, although there may be infinitely many k' for each $\mathbb{M}'$, a finite table suffices: For each machine $\mathbb{M}''$ with $|\mathbb{M}''| \leq n$ it stores the smallest k'' (if existent), such that $\mathbb{M}''$ halts in at most $|\mathbb{M}''|^{k''}$ steps.

If otherwise $m' > n$, we must have $k' \leq k$. Then the algorithm uses direct simulation to decide, whether $\mathbb{M}'$ halts in at most k' steps. This needs time

$$O(|\mathbb{M}'|^{k'}) \leq O(m^{c \cdot k'}) \leq O(m^{c \cdot k}).$$

Case 2: For all $k, n \in \mathbb{N}$ there is a Turing machine $\mathbb{M}$ such that $|\mathbb{M}'(\mathbb{M}, k)| > n$ and $k'(\mathbb{M}, k) > k$.

Then for all $k, n \in \mathbb{N}$,

$$g(k,n) \leq g(k'(\mathbb{M},k), |\mathbb{M}'(\mathbb{M},k)|) = \kappa'(R((\mathbb{M}, c \cdot k + 1))) \leq h(c \cdot k + 1),$$

which gives a computable bound on g in terms of its first argument k. Again, this is a contradiction. □

4 The classes $\mathscr{F}$-W[P]

We can bound the running time of $\mathscr{F}$-fpt algorithms by "fpt-clocks", that is by functions, which itself are computable by an $\mathscr{F}$-fpt algorithm. In particular, a parameterized problem (Q, κ) is in $\mathscr{F}$-fpt if there is a Turing machine $\mathbb{M}$ and a function $h : \Sigma^* \to \mathbb{N}$ such that

- $\mathbb{M}$ decides Q.
- On input $x \in \Sigma^*$ the machine $\mathbb{M}$ performs at most $h(x)$ steps.
- Using unary representation of the output, the function h is computable in $\mathscr{F}$-FPT time.

Recall the definition of the parameterized complexity class W[P]. We generalize it to arbitrary $\mathscr{F}$ thereby requiring that the running time and the bound on the number of nondeterministic steps are controlled by fpt-clocks.

Definition 29. Let $\mathscr{F}$ be a regular class of functions. A parameterized problem (Q, κ) over the alphabet Σ belongs to the class $\mathscr{F}$-W[P] if there is a nondeterministic Turing machine $\mathbb{M}$ and functions $h_1, h_2 : \Sigma^* \to \mathbb{N}$ with the following properties:

(1) $\mathbb{M}$ decides Q.

(2) On every run with input $x \in \Sigma^*$ the machine $\mathbb{M}$ performs at most $h_1(x)$ steps.

(3) On every run with input $x \in \Sigma^*$ the machine $\mathbb{M}$ performs at most $h_2(x) \cdot \log h_1(x)$ nondeterministic steps.

(4) Using unary representation of the output, the functions h_1 and h_2 are computable in $\mathscr{F}$-FPT time.

(5) There is a function $g \in \mathscr{G}(\mathscr{F})$ such that $h_2(x) \leq g(\kappa(x), |x|)$ for all $x \in \Sigma^*$.

Due to (2) and (4), in particular the running time of the machine $\mathbb{M}$ is an $\mathscr{F}$-FPT time.

A more intuitive definition of the class can be obtained if we use random access machines (RAMs) instead of Turing machines. We work with a standard RAM model: Registers are indexed by nonnegative integers; register 0 is the accumulator. Registers store nonnegative integers. The arithmetic operations are addition, subtraction (cut off at 0), and division by two (rounded off). We use a uniform cost measure. A nondeterministic RAM (NRAM) has an additional GUESS instruction whose semantics is: *Guess a natural number less than or equal to the number stored in the accumulator and store it in the accumulator.* (For details on the model, we refer the reader to the appendix of [9].) As random access machines only operate with nonnegative integers, we have to identify the letters of the input alphabet Σ of a decision problem with nonnegative integers, which in the following we do without mentioning it explicitly.

Let $h : \Sigma^* \to \mathbb{N}$. A deterministic or nondeterministic RAM-program $\mathbb{P}$ is *h-bounded*, if for every input $x \in \Sigma^*$ the program $\mathbb{P}$ on every run

- performs at most $h(x)$ steps;

- uses at most the first $h(x)$ registers;
- stores numbers less than or equal to $h(x)$ in any register at any time.

Let $\mathscr{F}$ be a class of unary functions and $\kappa : \Sigma^* \to \mathbb{N}$ a parameterization. A deterministic or nondeterministic RAM-program $\mathbb{P}$ is *$\mathscr{F}$-fpt bounded*, if it is h-bounded for some h, which is computable in $\mathscr{F}$-FPT time with respect to κ.

It is not hard to see that a parameterized problem is in W[P] if and only if it is decidable by a $\mathscr{C}$-fpt bounded program, which for some computable function $g : \mathbb{N} \to \mathbb{N}$, on any input in every run performs at most $g(k)$ nondeterministic steps [5]. Here we call a step *nondeterministic* if it carries out the guess instruction. We obtain a corresponding characterization of $\mathscr{F}$-W[P] for every regular class $\mathscr{F}$ of functions:

Proposition 30. *Let $\mathscr{F}$ be a regular class of functions. A parameterized problem (Q, κ) over the alphabet Σ is in $\mathscr{F}$-W[P] if and only if there is an NRAM-program $\mathbb{P}$ and functions $h_1, h_2 : \Sigma^* \to \mathbb{N}$ with the following properties:*

(1) $\mathbb{P}$ decides Q.

(2) $\mathbb{P}$ is h_1-bounded.

(3) For every input $x \in \Sigma^*$, in every run the program performs at most $h_2(x)$ nondeterministic steps.

(4) Using unary representation of the output, the functions h_1 and h_2 are computable in $\mathscr{F}$-FPT time.

(5) There is a function $g \in \mathscr{G}(\mathscr{F})$ such that $h_2(x) \leq g(\kappa(x), |x|)$ for all $x \in \Sigma^*$.

In particular, $\mathbb{P}$ is an $\mathscr{F}$-fpt algorithm.

Proof. Observe that guessing $h_2(x)$ numbers in the range $0, \ldots, h_1(x)$ amounts to guessing $O\big(h_2(x) \cdot \log h_1(x)\big)$ bits. The proof of the proposition is a straightforward simulation of Turing machines by random access machines and vice versa. □

The characterization of $\mathscr{F}$-W[P] given in Proposition 30 is "cleaner" than the definition in terms of Turing machines because it clearly separates the running time bound in terms of $\mathscr{F}$ from the bound on the number of nondeterministic steps in terms of $\mathscr{G}(\mathscr{F})$.

The following example gives a nice illustration of why W[P] is often viewed as the analogue of NP in the world of parameterized complexity theory (cf. Chap. 3 of [9]).

Example 31. $\mathscr{P}$-W[P] is essentially NP. More precisely, $\mathscr{P}$-W[P] is the class of all parameterized problems (Q, κ), where $Q \in$ NP. The same is true for all $\mathscr{F}$-W[P] for regular $\mathscr{F} \subseteq \mathscr{P}$.

Example 32. $2^{O(k)}$-W[P] is the class of all parameterized problems that can be solved by a $2^{O(k)}$-fpt NRAM-program with at most $O(k + \log n)$ nondeterministic steps.

Example 33. $2^{o(k)}$-W[P] is the class of all parameterized problems that can be solved by a $2^{o(k)}$-fpt NRAM-program with at most $O(k + \log n)$ nondeterministic steps. $2^{o^{\text{eff}}(k)}$-W[P] the class of all parameterized problems that can be solved by a $2^{o^{\text{eff}}(k)}$-fpt NRAM-program with at most $O(k + \log n)$ nondeterministic steps. Hence

$$2^{o^{\text{eff}}(k)}\text{-W[P]} \subseteq 2^{o(k)}\text{-W[P]} \subseteq 2^{O(k)}\text{-W[P]}.$$

Example 34. $\mathscr{A}$-W[P] is the class of all parameterized problems that can be solved by an NRAM-program, which running time and number of nondeterministic steps both are computable in deterministic $\mathscr{A}$-FPT time, and the latter is also bounded in terms of the parameter.

Example 35. We have

$$\text{W[P]} \subseteq \mathscr{C}\text{-W[P]} \subseteq \mathscr{A}\text{-W[P]}.$$

The first inclusion is trivial, and the second follows from Lemma 24.

Lemma 36. *For every regular class $\mathscr{F}$ of functions, $\mathscr{F}$-W[P] is downward closed under $\mathscr{F}$-fpt reducibility.*

Proof. Let (Q,κ) and (Q',κ') be parameterized problems over the alphabets Σ,Σ', respectively, such that $(Q,\kappa) \leq^{\mathscr{F}} (Q',\kappa')$ and $(Q',\kappa') \in \mathscr{F}$-W[P]. Let R be an $\mathscr{F}$-fpt reduction from (Q,κ) to (Q',κ'), and let $f \in \mathscr{F}, g \in \mathscr{G}(\mathscr{F})$ be witnesses for requirements (2) and (3) of Definition 11. Let the nondeterministic Turing machine $\mathbb{M}'$ and the functions $h'_1, h'_2 : (\Sigma')^* \to \mathbb{N}$ be witnesses for (Q',κ') being in $\mathscr{F}$-W[P]. Let f'_1, f'_2 be such that h'_1, h'_2 can be computed in time $f'_1(\kappa'(x')) \cdot |x'|^{O(1)}$, respectively $f'_2(\kappa'(x')) \cdot |x'|^{O(1)}$, from input x', and let g' be such, that $h'_2(x') \leq g'(\kappa'(x'), |x'|)$ for all $x' \in (\Sigma')^*$. Without loss of generality we may assume that the functions f, g, f'_1, f'_2, g' are nondecreasing.

Let $\mathbb{M}$ be a deterministic Turing machine computing R and let $h_3(x)$ denote the running time of $\mathbb{M}$ given input x. By composing $\mathbb{M}$ with $\mathbb{M}'$ we obtain a nondeterministic Turing machine $\mathbb{M}''$ that decides (Q,κ) in time

$$h''_1(x) := h_3(x) + h'_1(R(x)).$$

$h''_1(x)$ can be computed from x in time

$$h_3(x) + f'_1(g(\kappa(x),|x|)) \cdot (h_3(x))^{O(1)} \leq f(\kappa(x)) \cdot |x|^{O(1)} + f'_1(g(\kappa(x),|x|)) \cdot (f(\kappa(x)) \cdot |x|^{O(1)})^{O(1)}.$$

By the regularity of $\mathscr{F}$ and the definition of $\mathscr{G}(\mathscr{F})$ this is in $\mathscr{F}$-FPT time. Furthermore, $\mathbb{M}''$ uses at most $h''_2(x) \cdot \log h''_1(x)$ nondeterministic bits, where

$$h''_2(x) := h'_2(R(x)).$$

Similar to the above, h''_2 can be computed in $\mathscr{F}$-FPT time. It remains to show condition (5) of the definition. Here, we have

$$h''_2(x) = h'_2(R(x)) \leq g'(\kappa'(R(x)), |R(x)|) \leq g'(g(\kappa(x),|x|), f(\kappa(x)) \cdot |x|^{O(1)}).$$

By Lemma 10(2) and the definition of $\mathscr{G}(\mathscr{F})$, this is bounded by

$$g''(\kappa(x),|x|)$$

for some $g'' \in \mathscr{G}(\mathscr{F})$. □

Theorem 37. *p-WSAT(CIRC) is $\mathscr{F}$-W[P]-complete under $\mathscr{F}$-reductions.*

Proof. There is an NRAM-program $\mathbb{P}$ that decides whether a given circuit is k-satisfiable in polynomial time with at most k nondeterministic steps. By Lemma 10(4) and Proposition 30, this yields p-WSAT(CIRC) $\in \mathscr{F}$-W[P].

To see that p-WSAT(CIRC) is $\mathscr{F}$-W[P]-hard, let $(Q,\kappa) \in$ W[P] be arbitrary; we shall reduce (Q,κ) to p-WSAT(CIRC) by an $\mathscr{F}$-fpt reduction. Let $\mathbb{M}$, h_1, and h_2 be as in Definition 29. For input x, the reduction first computes $m = h_1(x)$ and $\ell = h_2(x)$, using $\mathscr{F}$-FPT time. By a standard simulation of $\mathbb{M}$ by circuits, we obtain a circuit C_x with $\ell \cdot \log m$ input nodes such that

$$x \in Q \iff C_x \text{ is satisfiable.}$$

Moreover, C_x can be constructed from x in time polynomial in ℓ and m and hence still by an $\mathscr{F}$-fpt algorithm. By applying the $k \cdot \log n$-trick (see [1] or Section 3.2 of [9]) to C_x, we get a circuit D_x of size $\text{poly}(\ell + m)$ with $O(\ell \cdot m)$ input nodes such that

$$C_x \text{ is satisfiable} \iff D_x \text{ is } \ell\text{-satisfiable.}$$

Altogether, we have

$$x \in Q \iff D_x \text{ is } \ell\text{-satisfiable.}$$

As $\ell \leq g(\kappa(x),|x|)$ for some $g \in \mathscr{G}(\mathscr{F})$, this yields an $\mathscr{F}$-reduction from (Q,κ) to p-WSAT(CIRC). □

Corollary 38. *Let $\mathscr{F}, \mathscr{F}'$ be regular and $\mathscr{F} \subseteq \mathscr{F}'$. Then*

$$\mathscr{F}\text{-W[P]} = \mathscr{F}\text{-FPT} \implies \mathscr{F}'\text{-W[P]} = \mathscr{F}'\text{-FPT}.$$

It follows that if PTIME = NP, then $\mathscr{F} \subseteq \mathscr{F}'$ implies

$$\mathscr{F}\text{-W[P]} = \mathscr{F}\text{-FPT} \subseteq \mathscr{F}'\text{-FPT} = \mathscr{F}'\text{-W[P]}$$

for all regular $\mathscr{F}, \mathscr{F}'$. On the other hand, if PTIME $\neq$ NP then $\mathscr{P}$-W[P] $\not\subseteq \mathscr{A}$-W[P]. Also note that (unconditionally), $\mathscr{A}$-W[P] $\not\subseteq \mathscr{P}$-W[P].

4.1 $\mathscr{F}$-W[P] *and Limited Nondeterminism*

The concept of limited nondeterminism was introduced by Kintala and Fischer [12]. The connection between parameterized complexity theory and limited nondeterminism has first been observed by Cai and Chen [2, 3]. Theorem 39 generalizes results of [4, 10].

Recall the definition of the "inverse" ι_f of a nondecreasing and unbounded function $f : \mathbb{N} \to \mathbb{N}$ (Definition 22). For a class $\mathscr{F}$ of unary functions, we let

$$\mathscr{I}(\mathscr{F}) = \{\iota_f \mid f \in \mathscr{F} \text{ nondecreasing and unbounded}\}.$$

For a function $h : \mathbb{N} \to \mathbb{N}$, we let

$$\mathrm{NP}[h]$$

(often, we write $\mathrm{NP}[h(n)]$) be the class of all (classical) problems that can be solved by a polynomially bounded nondeterministic Turing machine with binary branching that on inputs of length n performs at most $h(n)$ nondeterministic steps.

Theorem 39. *Let $\mathscr{F} \not\subseteq O(1)$ be an effectively regular class of unary functions. Then the following are equivalent:*

(1) $\mathscr{F}$-W[P] = $\mathscr{F}$-FPT.

(2) There is a $\iota \in \mathscr{I}(\mathscr{F})$ *such that*

$$\mathrm{NP}[\iota(n) \cdot \log n] = \mathrm{PTIME}.$$

Proof. For the direction from (1) to (2), assume that p-WSAT(CIRC) $\in \mathscr{F}$-FPT. Then there is a function $f \in \mathscr{F}$, a constant $c \in \mathbb{N}$, and an algorithm $\mathbb{A}$ that, given a circuit C and $k \in \mathbb{N}$ decides if $(C,k) \in p$-WSAT(CIRC) in at most $f(k) \cdot |C|^c$ steps. As $\mathscr{F}$ is regular, we can assume f to be time-constructible. We show that $\mathrm{NP}[\iota(n) \cdot \log n] \subseteq \mathrm{PTIME}$ for $\iota = \iota_f$.

Let $\mathbb{A}'$ be the algorithm that, on input (C,k), simulates $\mathbb{A}$ for at most $|C|^{c+1}$ steps and accepts if $\mathbb{A}$ accepts in at most $|C|^{c+1}$ steps and rejects otherwise. Observe that for inputs (C,k) with $k \leq \iota(|C|)$, the algorithm $\mathbb{A}$ halts in at most

$$f(k) \cdot |C|^c \leq f(\iota(|C|)) \cdot |C|^c \leq |C| \cdot |C|^c$$

steps and hence $\mathbb{A}'$ correctly decides if C is k-satisfiable.

Consider an arbitrary problem $Q \in \mathrm{NP}[\iota(n) \cdot \log n]$. Let $\mathbb{M}$ be a nondeterministic Turing machine deciding $x \in Q$ in polynomial time with at most $\iota(|x|) \cdot \log |x|$ nondeterministic steps. As f is time-constructible, we can compute $\ell := \iota(|x|)$ in polynomial time from x. As in the proof of Theorem 37, for every instance x of Q we can construct, in polynomial time, a circuit C_x with $\ell \cdot \log |x|$ input nodes such that

$$x \in Q \iff C_x \text{ is satisfiable.} \tag{4.1}$$

Applying the $k \cdot \log n$-trick again, from C_x we can construct a circuit D_x with $\Theta(\ell \cdot |x|)$ input nodes such that

$$C_x \text{ is satisfiable} \iff D_x \text{ is } \ell\text{-satisfiable.} \tag{4.2}$$

Then $|x| \leq |D_x|$, and as ι is nondecreasing, we have $\ell \leq \iota(|D_x|)$. We can therefore use the algorithm $\mathbb{A}'$ to decide whether D_x is ℓ-satisfiable in polynomial time. Thus $Q \in \mathrm{PTIME}$.

For the converse direction assume that $\mathrm{NP}[\iota(n) \cdot \log n] = \mathrm{PTIME}$ holds for $\iota = \iota_f$ for some nondecreasing and unbounded $f \in \mathscr{F}$. Since $\mathscr{F}$ is effectively regular, without loss of generality, we may assume that f is time-constructible. Then ι is polynomial time computable. It follows that there is a polynomial time algorithm $\mathbb{A}'$ that accepts precisely those instances (C,k) of p-WSAT(CIRC) for which $k \leq \iota(|C|)$ and C is k-satisfiable.

We show that p-WSAT(CIRC) $\in \mathscr{F}$-FPT. Let $\mathbb{A}$ be the following algorithm: Given an instance (C,k), it first checks if $k \leq \iota(|C|)$. If so, it simulates $\mathbb{A}'$ to decide if C is k-satisfiable. If $k > \iota(|C|)$, then algorithm $\mathbb{A}$ first constructs a circuit C' of size $\Omega(f(k) \cdot |C|)$ that is equivalent to C. This can be achieved, for example, by taking $f(k)$ disjoint copies of C, identifying their input nodes, and connecting their output nodes to a new

output node. Now $\mathbb{A}$ simulates $\mathbb{A}'$ on input (C',k). This yields the correct answer, because $k \leq \iota(f(k)) \leq \iota(|C'|)$. The running time of $\mathbb{A}$ can be bounded by

$$|C|^{O(1)} + O(f(k) \cdot |C|) + |C'|^{O(1)} \leq |C|^{O(1)} + f(k)^{O(1)} \cdot |C|^{O(1)},$$

which by the regularity of $\mathscr{F}$ is bounded by $f'(k) \cdot |C|^{O(1)}$. This proves that p-WSAT(CIRC) $\in \mathscr{F}$-FPT and hence that $\mathscr{F}$-W[P] $= \mathscr{F}$-FPT. □

The assumption $\mathscr{F} \not\subseteq O(1)$ in the theorem is only needed because the definition of ι_f required f to be unbounded, and hence $\mathscr{I}(O(1)) = \emptyset$. However, for $\mathscr{F} = O(1)$ we have the equivalence

$$O(1)\text{-W[P]} = O(1)\text{-FPT} \iff \text{NP} = \text{PTIME}.$$

The effectivity condition on $\mathscr{F}$ is necessary, as the following example shows:

Example 40. For the class $\mathscr{A}$ of all unary functions, $\mathscr{I}(\mathscr{A})$ is the class of all nondecreasing and unbounded functions. Let ι be nondecreasing and unbounded such that for every nondecreasing and unbounded computable function $h : \mathbb{N} \to \mathbb{N}$ there exists an n_0 such that for all $n \geq n_0$ it holds that $\iota(n) < h(n)$. Such a function ι can easily be constructed.

Note that for every polynomially bounded nondeterministic Turing machine $\mathbb{M}$, the function $h : \mathbb{N} \to \mathbb{N}$ defined by

$$h(n) = \max\big\{\ell \;\big|\; \text{there is an } x \text{ with } |x| \leq n \text{ such that on input } x, \text{ on some run the machine } \mathbb{M} \text{ performs exactly } \ell \text{ nondeterministic steps}\big\}$$

is computable. So, if $\mathbb{M}$ witnesses membership in NP$[\iota(n) \cdot \log n]$, then $\frac{h(n)}{\log n}$ must be bounded. Hence NP$[\iota(n) \cdot \log n] =$ PTIME, while it is generally believed that $\mathscr{A}$-W[P] $\neq \mathscr{A}$-FPT.

5 The classes $\mathscr{F}$-XP

Definition 41. Let $\mathscr{F}$ be a regular class of functions. A parameterized Problem (Q,κ) over the alphabet Σ belongs to the class $\mathscr{F}$-XP, if there are a Turing machine $\mathbb{M}$ and functions $h_1, h_2 : \Sigma^* \to \mathbb{N}$ with the following properties:

(1) $\mathbb{M}$ decides Q.

(2) On input $x \in \Sigma^*$ the machine $\mathbb{M}$ performs at most $h_1(x)^{h_2(x)}$ steps.

(3) Using unary representation of the output, the functions h_1 and h_2 are computable in $\mathscr{F}$-FPT time.

(4) There is a function $g \in \mathscr{G}(\mathscr{F})$ such that $h_2(x) \leq g(\kappa(x), |x|)$ for all $x \in \Sigma^*$.

Compare this to the definition of $\mathscr{F}$-W[P].

The following example illustrates why XP is often viewed as a parameterized analogue of exponential time.

Example 42. $O(1)$-XP $= \mathscr{P}$-XP is the class of all parameterized problems (Q,κ), where Q belongs to EXPTIME.

Example 43. $2^{O(k)}$-XP is the class of all parameterized problems, which can be solved in time $2^{O(k^2)} \cdot n^{O(\log n)}$, where n is the length of the input and k its parameter.

To see this, recall that $\mathscr{G}(2^{O(k)})$ is $O(k + \log n)$. Then it is immediate to obtain $2^{O((k+\log n)^2)}$ as a characterization of the running time. The claim follows from the standard inequality $(k + \log n)^2 \leq 4 \cdot (k^2 + (\log n)^2)$.

Example 44. $\mathscr{A}$-XP is the class of all parameterized problems (Q,κ), for which there are $h_1,h_2:\Sigma^*\to\mathbb{N}$, such that Q can be solved, given x, in time $h_1(x)^{h_2(x)}$, h_1 and h_2 are computable in $\mathscr{A}$-FPT time with respect to κ, and $h_2(x)$ is bounded in terms of $\kappa(x)$.

As $h_1(x)$ is implicitly bounded by $f(\kappa(x))+|x|^c$ for some $f\in\mathscr{A}$ and $c\in\mathbb{N}$, every problem (Q,κ) in $\mathscr{A}$-XP can be solved in particular in time $O(n^{f'(k)})$ for some function $f'\in\mathscr{A}$. It is an open question whether the converse holds, that is, whether all parameterized problems that can be solved in time $O(n^{f'(k)})$ for some function $f'\in\mathscr{A}$ are in $\mathscr{A}$-XP. We conjecture that this is not the case. However, observe that all parameterized problems that can be solved in time $O(n^{f''(k)})$ for some computable function $f''\in\mathscr{C}$ are in $\mathscr{A}$-XP. Thus XP $\subseteq\mathscr{A}$-XP.

It is easy to see, that $\mathscr{F}$-W[P] $\subseteq\mathscr{F}$-XP. Indeed, simply simulating the nondeterminism allowed for $\mathscr{F}$-W[P] by a deterministic exhaustive search yields an $\mathscr{F}$-XP running time.

Lemma 45. *Let $\mathscr{F}$ be regular. Then $\mathscr{F}$-*XP *is downward closed under $\mathscr{F}$-fpt reducibility.*

Proof. The proof is very similar to that of Lemma 36

Let (Q,κ) and (Q',κ') be parameterized problems over the alphabets Σ,Σ', respectively, such that $(Q,\kappa)\leq^{\mathscr{F}}(Q',\kappa')$ and $(Q',\kappa')\in\mathscr{F}$-XP. Let R be an $\mathscr{F}$-fpt reduction from (Q,κ) to (Q',κ'), and let $f\in\mathscr{F},g\in\mathscr{G}(\mathscr{F})$ be witnesses for requirements (2) and (3) of Definition 11. Let the Turing machine $\mathbb{M}'$ and the functions $h_1',h_2':(\Sigma')^*\to\mathbb{N}$ be witnesses for (Q',κ') being in $\mathscr{F}$-XP. Let f_1',f_2' be such, that h_1',h_2' can be computed in time $f_1'(\kappa'(x'))\cdot|x'|^{O(1)}$, respectively $f_2'(\kappa'(x'))\cdot|x'|^{O(1)}$ and let g' be such, that $h_2'(x')\leq g'(\kappa'(x'),|x'|)$ for all $x'\in(\Sigma')^*$. Without loss of generality we may assume that the functions f,g,f_1',f_2',g' are nondecreasing.

Let $\mathbb{M}$ be a Turing machine computing R and let $h_3(x)$ denote the running time of $\mathbb{M}$ given input x. By composing $\mathbb{M}$ with $\mathbb{M}'$ we obtain a Turing machine $\mathbb{M}''$ that decides (Q,κ). By letting $h_1''(x):=h_3(x)+h_1'(R(x))$ and $h_2''(x):=h_2'(R(x))$, we claim that $\mathbb{M}''$, h_1'', and h_2'' are witnesses for (Q,κ) being in $\mathscr{F}$-XP. The running time of $\mathbb{M}''$ on input $x\in\Sigma^*$ is bounded by

$$h_3(x)+(h_1'(R(x)))^{h_2'(R(x))}\leq(h_3(x)+h_1'(R(x)))^{h_2'(R(x))}=(h_1''(x))^{h_2''(x)}.$$

Further note, that h_1'' and h_2'' are defined exactly as in the proof of Lemma 36. There it has already been shown, that they fulfill requirements (4) and (5) of Definition 29, which are identical to requirements (3) and (4) of Definition 41. All together prove the claim. □

The technically most important problem in $\mathscr{F}$-XP is the following:

p-EXP-DTM-HALT
Instance: A Turing machine $\mathbb{M}$ and $k\in\mathbb{N}$.
Parameter: k.
Problem: Decide, whether $\mathbb{M}$ halts on empty input in at most $\lvert\mathbb{M}\rvert^k$ steps.

Theorem 46. *For all regular $\mathscr{F}$, the problem p-*EXP-DTM-HALT *is complete for $\mathscr{F}$-*XP.

Proof. Using Lemma 10(4), we obtain p-EXP-DTM-HALT $\in\mathscr{F}$-XP by a direct simulation of the input machine $\mathbb{M}$ for $|\mathbb{M}|^k$ steps.

For hardness, let (Q,κ) be a problem in $\mathscr{F}$-XP and let $\mathbb{M}$, h_1, and h_2 be witnesses for this. The reduction from (Q,κ) to p-EXP-DTM-HALT works as follows: On input $x\in\Sigma^*$ it first computes $N:=h_1(x)$ and $K:=h_2(x)$. Then it rewrites $\mathbb{M}$ to $\mathbb{M}_x$ in the following way:

(1) Instead of accepting or rejecting, $\mathbb{M}_x$ halts or loops.

(2) $\mathbb{M}_x$ starts by writing x to the input tape before simulating $\mathbb{M}$. This can be done using $O(|x|)$ additional steps at the beginning.

(3) $\mathbb{M}_x$ is padded so that $|\mathbb{M}_x|\geq O(|x|)+N$.

The reduction outputs $(\mathbb{M}_x,K)$. □

Theorem 47. *For each regular class $\mathscr{F}$ of functions, $\mathscr{F}$-FPT $\neq \mathscr{F}$-XP.*

Proof. Otherwise, in particular p-EXP-DTM-HALT $\in \mathscr{F}$-FPT. Then, let $f \in \mathscr{F}$ and c be such, that p-EXP-DTM-HALT can be solved on input $(\mathbb{M},k)$ in time $f(k)\cdot(|\mathbb{M}|+k)^c$. For fixed $k > c$ this contradicts the time hierarchy theorem. □

The definition of $\mathscr{F}$-XP is more complicated than that of XP. Except for its Requirement (3), this was necessary to make Lemma 45 true. Requirement (3) was necessary, to have Theorem 46. One might hope, that at least $\mathscr{C}$-XP nevertheless coincides with XP. However, independent from Requirement (3), this is not the case.

Proposition 48. XP *is not downward closed under $\mathscr{C}$-FPT-reducibility. In particular,* XP $\neq \mathscr{C}$-XP.

Proof. Let $g : \mathbb{N}^2 \to \mathbb{N}$ be as in Lemma 27. Consider the following parameterized problem:

p-EXP-DTM-HALT$'$
Instance: A Turing machine $\mathbb{M}$ and $k \in \mathbb{N}$.
Parameter: k.
Problem: Decide, whether $\mathbb{M}$ halts on empty input in at most $\lvert\mathbb{M}\rvert^{g(k,\lvert\mathbb{M}\rvert)}$ steps.

p-EXP-DTM-HALT belongs to XP and by virtue of definition we have

$$p\text{-EXP-DTM-HALT}' \leq^{\mathscr{C}} p\text{-EXP-DTM-HALT}.$$

Now, for contradiction, assume p-EXP-DTM-HALT$' \in$ XP. Then there is a computable function $f : \mathbb{N} \to \mathbb{N}$, such that p-EXP-DTM-HALT$'$ can be decided, given $(\mathbb{M},k)$, in time $(|M|+k)^{f(k)}$. Now let k_0, n_0 be such, that $c := f(k_0) < g(k_0,n_0)$. As g is nondecreasing, we also have $c+1 \leq g(k_0,n)$ for all $n \geq n_0$. Thus, using the XP algorithm, we can simulate $O(n^{c+1})$ time using $O(n^c)$ time, which contradicts the time hierarchy theorem. □

References

[1] K.A. Abrahamson, R.G. Downey, and M.R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogs. *Annals of Pure and Applied Logic*, 73:235–276, 1995.

[2] L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *Journal of Computer and System Sciences*, 54:465–474, 1997.

[3] L. Cai and J. Chen. On the amount of nondeterminism and the power of verifying. *SIAM Journal on Computing*, 26(3):733–750, 1997.

[4] L. Cai, J. Chen, R.G. Downey, and M.R. Fellows. On the structure of parameterized problems in NP. *Information and Computation*, 123:38–49, 1995.

[5] Y. Chen, J. Flum, and M. Grohe. Machine-based methods in parameterized complexity theory. *Theoretical Computer Science*, 339:167–199, 2005.

[6] Y. Chen and M. Grohe. An isomorphism between subexponential and parameterized complexity theory. In *Proceedings of the 21st Conference on Computational Complexity*, pages 314–328, 2006.

[7] N.J. Cutland. *Computability*. Cambridge University Press, 1980.

[8] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[9] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[10] J. Flum, M. Grohe, and M. Weyer. Bounded fixed-parameter tractability and $\log^2 n$ nondeterministic bits. *Journal of Computer and System Sciences*, 72:34–71, 2006.

[11] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130:3–31, 2004.

[12] C. Kintala and P. Fischer. Refining nondeterminism in relativised polynomial time bounded computations. *SIAM Journal on Computing*, 9:46–53, 1980.

[13] N. Robertson and P.D. Seymour. Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

[14] M. Weyer. Bounded fixed-parameter tractability: The case $2^{\text{poly}(k)}$. In R.G. Downey, M. Fellows, and F. Dehne, editors, *Proceedings of the 1st International Workshop on Parameterized and Exact Computation*, volume 3162 of *Lecture Notes in Computer Science*, pages 49–60. Springer-Verlag, 2004.