Uniform Characterizations of Complexity Classes of Functions*

Sven Kosub *Heinz Schmitz*** *Heribert Vollmer*
Theoretische Informatik, Universität Würzburg
Am Hubland, D-97074 Würzburg, Germany
{*kosub,schmitz,vollmer*}*@informatik.uni-wuerzburg.de*

Abstract

We introduce a general framework for the definition of function classes. Our model, which is based on nondeterministic polynomial-time Turing transducers, allows uniform characterizations of FP, $\mathrm{FP}^{\mathrm{NP}}$, $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$, $\mathrm{FP}^{\mathrm{NP}}_{\mathrm{tt}}$, counting classes ($\#\cdot\mathrm{P}$, $\#\cdot\mathrm{NP}$, $\#\cdot\mathrm{coNP}$, GapP, $\mathrm{GapP}^{\mathrm{NP}}$), optimization classes (max·P, min·P, max·NP, min·NP), promise classes (NPSV, $\#_{\mathrm{few}}\cdot\mathrm{P}$, $\mathrm{c}\#\cdot\mathrm{P}$), multivalued classes (FewFP, NPMV), and many more. Each such class is defined in our model by a scheme how to evaluate computation trees of nondeterministic machines. We study a reducibility notion between such evaluation schemes, which leads to a necessary and sufficient criterion for relativizable inclusion between function classes. As it turns out, this criterion is easily applicable and we get as a consequence, e.g., that there is an oracle A, such that $\min\cdot\mathrm{P}^A \not\subseteq \#\cdot\mathrm{NP}^A$ (note that no structural consequences are known to follow from the corresponding positive inclusion).

1 Introduction

Starting with the work of Valiant [35], Krentel [24, 25], and Toda [33], complexity classes of functions have gained considerable interest. Krentel showed that considering functional problems in their original form (e.g., "compute the length of the shortest traveling salesperson tour") instead of artificially coding them into yes-no-problems ("is the shortest traveling salesperson tour shorter than k?") allows one to make finer distinctions between different NP-complete problems than known previously. Toda showed that the class #P defined by Valiant has an unexpected power: every set from the polynomial hierarchy is Turing reducible to some #P function. Thus, to consider function classes turns out to be a worthwhile study.

But how to compare function classes, i.e., how to establish inclusions between them? And how do function classes relate to set classes? For this end—also going back to Toda's seminal paper [33] (see also [13])—operators have been used widely. Vollmer and Wagner [39, 40] introduced a pair of operators establishing a one-one correspondence between Krentel's maximization classes [25] and the classes that form the polynomial hierarchy; this gives the result that two

*A preliminary version of this paper appeared as [23].
**Supported by the Deutsche Forschungsgemeinschaft (DFG), grant Wa 847/1-2.

classes of Krentel's hierarchy collapse if and only if the corresponding classes of the Meyer-Stockmeyer hierarchy collapse. These results have been pushed a bit further in [37, 30, 14]. The general framework for the concept that in recent literature has been called the *operator method to separate function classes* can be described as follows: Given two function classes $\mathcal{F}_1$ and $\mathcal{F}_2$ and a monotonic operator $\mathcal{O}$ transforming them into set classes $\mathcal{K}_1$ and $\mathcal{K}_2$ (i.e., $\mathcal{F}_1 \subseteq \mathcal{F}_2$ implies $\mathcal{K}_1 \subseteq \mathcal{K}_2$), the following is clear: If we knew $\mathcal{K}_1 \neq \mathcal{K}_2$ then we would have proved $\mathcal{F}_1 \neq \mathcal{F}_2$. Unfortunately, results of the form $\mathcal{K}_1 \neq \mathcal{K}_2$ are very rare in computational complexity theory. But of course under the above circumstances, if we have evidence against $\mathcal{K}_1 \subseteq \mathcal{K}_2$, then this also serves as evidence against $\mathcal{F}_1 \subseteq \mathcal{F}_2$. For example, if $\mathcal{F}_1, \mathcal{F}_2, \mathcal{K}_1, \mathcal{K}_2$ have a reasonable model of oracle access and if $\mathcal{O}$ is a relativizable operator with respect to this model (i.e., if $\mathcal{O}$ transforms $\mathcal{F}$ into $\mathcal{K}$ then it also transforms $\mathcal{F}^A$ into $\mathcal{K}^A$ for any oracle A), then we conclude that an oracle separating $\mathcal{K}_1$ from $\mathcal{K}_2$ will also separate $\mathcal{F}_1$ from $\mathcal{F}_2$. Along these lines a lot of structural results were obtained in the above mentioned papers.

However, this approach sometimes fails. A prominent example is the case of the classes $\#\cdot$NP and min$\cdot$P, left as an open problem in [14]. Since this will become important later, let us look a bit closer at the background of this problem.

Valiant's above mentioned class #P [34] is the class of functions that count accepting paths of nondeterministic polynomial-time Turing machines: A function f belongs to #P iff there is such a machine M such that for all x, we have that $f(x)$ is the number of accepting paths of M on input x. Let us assume that the computation tree of M is balanced, where we adopt the definition of *balanced* from [18], which requires that given a number i, in polynomial time it can be determined if M has at least i paths and if so, the sequence of nondeterministic choices that M makes on the i-th path during its computation on x is computable in polynomial time. From this sequence then, the result of the i-th path of M on input x is again computable in polynomial time. (Note that in the context of the class #P, this requirement can be assumed without loss of generality.) Hence, we conclude that if we define A to consist of those pairs (x, y) where the y-th path of M on input x is accepting, we obtain $A \in$ P. Moreover it follows that the number of paths of M on inputs of length n is given by some polynomial-time computable function. Hence, a function f belongs to #P if and only if there exist a set $A \in$ P and a function $g \in$ FP (the class of polynomial-time computable functions) such that for all x, we have

$$f(x) = \left\| \{\, y \mid y \leq g(x) \wedge (x, y) \in A \,\} \right\|, \tag{1}$$

where $\|S\|$ for a set S denotes its cardinality.

The class span-P (defined in [21]) is the class of functions counting the number of different outputs of nondeterministic polynomial-time Turing machines, where every possible computation path may produce an output string. These machines are sometimes also called transducers; hence a function f belongs to span-P iff there is a nondeterministic polynomial-time Turing transducer M such that for all x, we have that $f(x)$ is the number of different outputs over all paths in the computation tree of M on input x. Interestingly, span-P has a characterization similar to the one of #P in (1) above. It can be shown that a function f belongs to span-P if and only if there exist a set $A \in \mathrm{NP}$ and a function $g \in \mathrm{FP}$ such that for all x, we have $f(x) = \big\|\{\, y \mid y \leq g(x) \wedge (x,y) \in A \,\}\big\|$. This characterization is sometimes denoted by the equality span-P $= \#\cdot\mathrm{NP}$, referring to the fact that here we have $A \in \mathrm{NP}$ instead of $A \in \mathrm{P}$ as in the case of #P, also denoted by $\#\cdot\mathrm{P}$, in (1) above.

Using a very similar definition pattern, the optimization classes $\max\cdot\mathrm{P}$ and $\min\cdot\mathrm{P}$ were introduced in [14]. The difference lies in the replacement of counting (cardinality) by the operation of maximization or minimization. The class $\max\cdot\mathrm{P}$ is defined to consist of those functions f for which there are a set $A \in \mathrm{P}$ and a function $g \in \mathrm{FP}$ such that for all x, we have

$$f(x) = \max\{\, y \mid y \leq g(x) \wedge (x,y) \in A \,\}. \tag{2}$$

The class $\min\cdot\mathrm{P}$ is defined analogously with min instead of max. Why are these classes interesting and deserve study? Clearly, $\max\cdot\mathrm{P} \subseteq \mathrm{MaxP}$ and $\min\cdot\mathrm{P} \subseteq \mathrm{MinP}$, where MaxP and MinP are Krentel's classes determining the maximal (minimal, resp.) output produced by nondeterministic polynomial-time Turing transducers [24]. However, there are problems in MaxP, e.g., the maximal satisfying assignment problem, which are known to be metrically complete even for $\mathrm{FP}^{\mathrm{NP}}$, but do not need the full computational power of MaxP. In fact, the mentioned problem lies in the presumably smaller class $\max\cdot\mathrm{P}$. Stated in other words, the maximal satisfying assignment problem is in MaxP but is not complete for this class under *functional many-one reductions* [37] (unless, as all these classes are closed under functional many-one reductions, $\max\cdot\mathrm{P} = \mathrm{MaxP}$; this latter collapse is equivalent to $\mathrm{P} = \mathrm{NP}$). It even turned out that $\mathrm{MaxP} = \max\cdot\mathrm{NP}$ and $\mathrm{MinP} = \min\cdot\mathrm{NP}$, where these classes are defined as in (2) but replacing the requirement $A \in \mathrm{P}$ by $A \in \mathrm{NP}$.

In general, counting classes are more powerful than optimization classes: we have the inclusions $\max\cdot\mathrm{NP} \subseteq \#\cdot\mathrm{NP}$ and $\min\cdot\mathrm{NP} \subseteq \#\cdot\mathrm{P}^{\mathrm{NP}}$, and it is known that these inclusions are in a sense optimal, i.e., we know that $\min\cdot\mathrm{NP} \not\subseteq \#\cdot\mathrm{NP}$ (unless $\mathrm{NP} = \mathrm{coNP}$) and $\#\cdot\mathrm{NP} \not\subseteq \min\cdot\mathrm{NP}$ (unless $\mathrm{NP} = \mathrm{PP}$). What about the subclass $\min\cdot\mathrm{P}$? Clearly $\min\cdot\mathrm{P} \subseteq \mathrm{MinP} \subseteq \#\cdot\mathrm{P}^{\mathrm{NP}}$, and by the above $\#\cdot\mathrm{NP} \not\subseteq \min\cdot\mathrm{P}$ (unless $\mathrm{NP} = \mathrm{PP}$), but the inclusion $\min\cdot\mathrm{P} \subseteq \#\cdot\mathrm{NP}$ is open. Here, the operator method did not lead to success so far. All known operators transform $\min\cdot\mathrm{P}$ and $\#\cdot\mathrm{NP}$ into set classes $\mathcal{K}_1$ and $\mathcal{K}_2$, resp., such that $\mathcal{K}_1 \subseteq \mathcal{K}_2$ (even relativizably). Thus it turns out that it can be very difficult to find a suitable operator $\mathcal{O}$. One might argue that the failure to find such an operator shows that the inclusion $\min\cdot\mathrm{P} \subseteq \#\cdot\mathrm{NP}$ is likely to hold, but no proof of this is known. It seems that a simple transformation to the set side does not help in this case.

In the present paper we establish a new way to compare function classes without reducing them to set classes. We introduce a uniform framework for the definition of function classes. This is done by using one single computation model (nondeterministic polynomial-time Turing transducers) and considering different evaluation schemes for this model. Our approach is motivated by the use of leaf languages as a uniform vehicle for the definition of *set classes*. Again, look at nondeterministic polynomial-time Turing machines. These produce, given an input word x, a computation tree with exponentially many paths (in the input length). Suppose that accepting paths output the symbol 1 while rejecting paths output 0. The sequence of all these output symbols in the order of paths (induced by the order of nondeterministic choices of M) is the so called *leaf string* of M on x. For a language A, the strings accepted by M with respect to A are exactly those strings x such that the leaf string produced by M on input x is in A. The class of all languages acceptable in this way is denoted by (A)-P. The language A is called the leaf language for this class. As an example, let us look at the class NP. By definition, a language $L \in \mathrm{NP}$ is given by a nondeterministic polynomial-time Turing machine M such that for all inputs x, we have that x belongs to L if and only if in the computation tree that M produces when working on x we find at least one accepting path. Hence, NP is defined by the leaf language $0^*1(0+1)^*$. A language $L \in \oplus\mathrm{P}$ [28] is given by a nondeterministic polynomial-time Turing machine M such that for all inputs x, we have that x belongs to L if and only if in the computation tree of M on x the number of accepting paths is even: $\oplus\mathrm{P} = \big((0^*10^*1)^*0^*\big)$-P. This framework, generalized to the case of machines that output symbols from an arbitrary alphabet (not necessarily the binary alphabet), was developed by Papadimitriou and Sipser around 1979 while teaching a course on complexity at MIT [27]. It was later rediscovered and published independently in [9, 36] and has since then been used actively in the study of complexity classes mostly in between NC^1 and PSPACE, see [16, 5, 15, 17, 6, 18, 7].

In a very similar way to the leaf language approach to define classes of sets, a function class in our framework is given by specifying how the computation of a nondeterministic polynomial-time Turing *transducer* is evaluated to compute the function value. We give this specification using so called *generators*. A generator is nothing else than a function which, from the sequence of outputs of the transducer, determines the result of the overall computation. Turning back to #P, let $f \in \#\mathrm{P}$ count the number of accepting paths of a nondeterministic polynomial-time Turing machine M. Looking at M as a transducer which outputs 1 on accepting paths and 0 on rejecting paths, we see that the value $f(x)$ is the sum of all outputs of M on input x. Hence, the generator that defines #P is summation. In the case of MinP (yielding the minimal output of a transducer) it is simply the operation of minimization. Strictly speaking, a generator is given not only by a function as just described, but also by specifying the allowed output sequences of our transducers. We will see that, when we restrict the set of allowed sequences, we are able to capture promise classes.

Almost all up-to-now considered classes of functions arising in the polynomial-time context are definable in our way. We think that using one single computation model for the definition of different classes may make reasoning about them and comparing them to one another easier (see also the discussion in [38]). It may make clear similarities and differences that otherwise are hidden in the peculiarities of particular models.

Next we study the inclusions between function classes not by relating them to inclusions among set classes, as in the operator method described above, but by comparing the power of their generators. This comparison will be done in the form of a suitable reducibility, and we show that if one generator reduces to a second one, then the class defined by the first generator is included (relativizably) in the class defined by the second generator. The main technical contribution of the present paper then is a necessary and sufficient criterion for the separability of function classes by oracles. We show that generator F_1 does not reduce to generator F_2 if and only if there is an oracle separating the function class defined by F_1 from the class defined by F_2. As we will see, our criterion is easily applicable, and we use it to attack (among others) the above mentioned open question: We prove the existence of an oracle A such that $\min\cdot \mathrm{P}^A \not\subseteq \#\cdot \mathrm{NP}^A$, a result also achieved by Glaßer and Wechsung [12].

The above mentioned criterion for oracle separations is a generalization of corresponding theorems for classes of sets, given in [9, 36]. In fact, our Theorem 5.1 gives the main result from these papers as a corollary. In a similar way as done for set classes in [9, 36], Theorem 5.1 we give below proves once and for all the diagonalization part of oracle constructions for function classes. Thus when applying our theorem to prove a separation, one no longer has to worry about the cumbersome details of a stage construction but can concentrate solely on the combinatorial problems.

The status of oracle separations of course is somewhat questionable, since we know of separations also in cases where the unrelativized classes collapse. However, it is widely accepted that oracle results can at least help to direct research, see, e.g., [1, 11]. We believe that this is particularly true in the case of a uniform computation model as we use it here, and we hope that oracle separations still may contribute to our knowledge about the complexity of the real (i.e., unrelativized) problem.

A similar approach to the one presented here for the definability of total, non-promise function classes was suggested by Borchert in [4]. More precisely, his evaluation of computation trees is not based on the sequence of leaf values, but considers the whole tree. In his framework however, no result allowing comparison of classes in form of these evaluation schemes is known. In contrast, we will below examine inclusions between function classes based on a comparison of their generators.

2 Preliminaries

We use the alphabet $\Sigma = \{0,1\}$. As usual we identify Σ^+ with the set of natural numbers $\mathbb{N} = \{0,1,2,\dots\}$ by binary encoding. For a number $m \in \mathbb{N}$ let $\mathrm{bin}(m)$ denote its binary encoding without leading zeroes. For a zero-one string x, where we allow leading zeroes, let $\mathrm{val}(x)$ denote the number it represents in binary (when deleting leading zeroes). Clearly, for $m \in \mathbb{N}$, we have $m = \mathrm{val}\big(\mathrm{bin}(m)\big)$. We use the order on $\mathbb{N}$ to define an order on Σ^* by $\varepsilon < x$ for $x \neq \varepsilon$, where ε denotes the empty word, and $x \leq y$ if $\mathrm{val}(x) < \mathrm{val}(y)$ or $\mathrm{val}(x) = \mathrm{val}(y) \wedge |x| \leq |y|$ for $x, y \neq \varepsilon$. For $\ell, r \in \mathbb{N}$, we denote the corresponding integer interval by $[\ell, r] =_{\mathrm{def}} \{\ell, \ell+1, \dots, r\}$. For a set S, let $\|S\|$ denote its cardinality.

Let Ω be the set of all finite vectors (sequences) of elements of Σ^+. For $\vec{u} \in \Omega$, $\vec{u} = (u_1, \dots, u_k)$, the dimension of $\vec{u}$ is defined as $\dim \vec{u} =_{\mathrm{def}} k$. Given a function $s\colon \mathbb{N} \to \mathbb{N}$, let Ω_s denote the set of all vectors $\vec{u}$ such that all components of $\vec{u}$ are (in value) less-than or equal to $s(\dim \vec{u})$, i.e., $\Omega_s =_{\mathrm{def}} \bigcup_{k\geq 1} \big\{\, x \;\big|\; \mathrm{val}(x) \leq s(k) \,\big\}^k$. Observe that if we use a standard encoding of sequences of binary words, then Ω can be identified with Σ^+. We define the length $|\vec{u}|$ of $\vec{u}$ as the sum of the length of its components, i.e., $|\vec{u}| =_{\text{def}} \sum_{i=1}^{k} |u_i|$.[1] Observe that $\dim \vec{u} \leq |\vec{u}|$ and, for $\vec{u} \in \Omega_s$, $|\vec{u}| \leq \dim \vec{u} \cdot \log s(\dim \vec{u})$. These easy inequalities will be very helpful later to determine resource requirements, since it will often be easier to estimate resources as a function of $\dim \vec{u}$ instead of $|\vec{u}|$.

In this paper, we do not require that all our functions are total, but also allow partial functions. The reason for this is that we want to make our framework also amenable to the examination of partial (multi-valued) function classes from the literature, such as NPSV or NPMV [31]. When we require that a function is total, this will be explicitly remarked. For functions φ and ψ, let $\varphi(x) \simeq \psi(x)$ denote that $\varphi(x)$ is defined if and only if $\psi(x)$ is defined, and if $\varphi(x)$ is defined then $\varphi(x) = \psi(x)$. We use the notation $\varphi \simeq \psi$ to denote that, for all inputs x, $\varphi(x) \simeq \psi(x)$. (Clearly, this is nothing else than equality of functions, but we will use the special symbol $\simeq$ to remind the reader that we are talking about partial functions.) The domain of a partial function φ is denoted by D_φ. For a function φ, $\varphi|_M$ denotes the restriction of φ to M, i.e., the function that coincides with φ on M and is undefined outside of M.

For total, number-theoretic functions f and g, let $f \geq_{a.e.} g$ denote that f is greater than or equal to g almost everywhere, i.e., there is some n_0 such that for all $n \geq n_0$ we have $f(n) \geq g(n)$. Let id denote the identity function given by $id(n) =_{\text{def}} n$ for all $n \in \mathbb{N}$.

We assume the reader to be familiar with basic complexity classes and reducibilities. Refer to the standard literature, e.g., [8, 26, 2]. Let FP be the class of total functions computed in polynomial time by deterministic Turing transducers. With FP^α we denote the class of functions computed in polynomial time by deterministic oracle Turing transducers with total oracle function α. An oracle machine is equipped with an oracle query tape, on which it writes the intended query x, and an oracle answer tape, on which it receives $\alpha(x)$ in one step. We assume that the query tape is not erased by this operation (though, of course, this point is not critical, since we are dealing with polynomial-time computations).

To investigate sub-linear time reductions, we use transducers with two particularities. First, in order to realize index access to their input, these machines have besides their work-tapes some special tapes: an index tape and an input tape. With i written on the index tape, the i-th argument of the machine's input vector appears in one step on the input tape, i.e., if the input is $\vec{u} = (u_1, \ldots, u_k)$ then the machine receives in one step the whole element u_i.[2] The index tape is not erased by this operation (this is the so called *unrestricted mode*; background on these machines can be found in [29]). Finding a blank symbol on the input tape tells the

[1] We note that the "standard" definition of the length of $\vec{u}$ as the length over Σ with respect to the particular encoding of sequences of Σ^+ does not lead us to different classes in our context below, i.e., for polylogarithmic time bounds and above.

[2] We should remark that our model admits the case that a machine receives an element u_i which has more symbols than the time-bound of M permits the machine to read. Seemingly, a model with two index tapes, where the machine, with i and j on these tapes, receives the jth symbol of u_i, is more natural. However, we remark that for all function classes we consider in this paper the entries in all vectors $\vec{u}$ will be relatively short such that the unwanted situation above does not occur. Therefore, we choose to stick to our simple model with only one index tape.

machine that there was no i-th argument. On a third special tape the machine gets as input the length of its input vector. Second, the transducers have special output tapes different from the input tapes. The outputs computed by a transducer are the tape inscriptions on its output tape. It is crucial that in this model one cannot copy the input (vector) from the input tape to the output tape in sub-linear time. This implies that under sub-linear time restrictions the identity function (for vectors) cannot be computed. Let FPLT denote the class of functions computable by such transducers in polylogarithmic time (in the length of the input vector).

3 How to Define Function Classes

3.1 The General Framework

In order to emphasize the essential characteristics of different function classes we introduce the general notion of a generator. A *generator* consists of a set $\Phi \subseteq \Omega$ of *admissible sequences* together with a function F over Ω that *evaluates* those sequences. Every generator (F, Φ) defines the class (F, Φ)-FP as the class of functions f for which there exist two total polynomial-time computable functions g and h such that for all x, the sequence $S_x = \big(g(x, 0), g(x, 1), \ldots, g(x, h(x))\big)$ belongs to Φ and $f(x) = F(S_x)$.

For example, let M be a nondeterministic Turing machine such that the computation tree of M is always balanced, and let $h \in$ FP compute the number of computation paths of M on input x. Consider the function g such that $g(x, i)$ is determined by first computing from i the nondeterministic choices of M on its i-th path and then simulating M with these choices, where we finally let $g(x, i) = 1$ if this computation accepts, and $g(x, i) = 0$ otherwise. With Φ the set of all finite vectors over $\{0, 1\}$ and F the operation of summing (values of) elements of a sequence, (F, Φ)-FP thus captures all of $\#\cdot$P. It is interesting to see that every other choice of $g, h \in$ FP also yields a function in $\#\cdot$P, hence $\#\cdot$P $= (F, \Phi)$-FP.

As another example, let us consider the class GapP of functions giving the number of accepting paths minus the number of rejecting paths of nondeterministic Turing machines. Remarkably GapP can be characterized using the same F as above but now taking Φ as the set of all finite vectors over $\{-1, 1\}$ (strictly speaking, the set of all finite vectors of encodings of these values over Σ^+). In this case, F does not map into $\mathbb{N}$ but into the set of the integers $\mathbb{Z}$. Below, we will also consider operations F that map into the set of all subsets of $\mathbb{N}$.

So we see that our notion of generators is directly connected with the idea of evaluating computation trees of nondeterministic polynomial time transducers, similar to the case of leaf languages for classes of sets. Considering $\Phi \neq \Omega$ allows us to capture promise classes, since we may in this way exclude certain machines from the definition of our class. The class GapP is of course not a promise class, though we just characterized it with a generator where $\Phi \neq \Omega$. Below, we will present another characterization for GapP, which will more directly reflect that it is not a promise class.

In the upcoming sections, we will study relativized versions (F,Φ)-FP^{α} for some oracle function α, defined similarly by now choosing g and h from FP^{α}.

So far, generators are a very general notion. As it turns out, the generators we use in the following all have a special property which we call *paddability*. Let us say that the generator (F,Φ) is *paddable*, if there is a constant $\gamma \in \Sigma^{+}$ such that for every $k \geq 1$ and for every $\vec{u} = (u_1,\ldots,u_k) \in \Omega$ it holds that

1. $F(u_1,\ldots,u_k) \simeq F(u_1,\ldots,u_k,\gamma)$,
2. $\vec{u} \in \Phi$ if and only if $(\vec{u},\gamma) \in \Phi$.

In this case, γ is called a *neutral element.*

As an example, if we have a function from $\#\cdot\mathrm{P}$ given by some nondeterministic polynomia-time Turing machine M, then it does not change the value of the function if we add to M some rejecting paths, i.e., the generator for $\#\cdot\mathrm{P}$ is paddable with neutral element 0.

3.2 Examples for Characterizations of Classes

In Tables 1 to 4 we list more function classes with their characterizing generators. Our strategy is to evaluate the outputs of a nondeterministic computation tree with a suitable generator and to capture the computation tree itself with help of the FP functions g,h from the generator definition. Little adjustments will be needful to provide paddability. The neutral element γ can be fixed as 0 in these tables except for $\min\cdot\mathrm{P}$, for which any string γ with $\mathrm{val}(\gamma) \geq 1$ can be taken. For the notations in the tables, let on input $\vec{u} = (u_1,\ldots,u_k)$ the indices range from 1 to k, and let $\mathrm{span}_{+}(\vec{u})$ be the cardinality of the set of nonzero components of $\vec{u}$. For simplicity of presentation, we do not distinguish in the tables between words over Σ and natural numbers, i.e., we omit the explicit mention of the functions bin and val.

We recall the definitions of some function classes in order to exemplify our characterizations and start with a formal definition of the operators already mentioned in the introduction. The operator $\#\cdot$ was introduced in [32] and generalizes Valiant's class #P [34]. The operators $\max\cdot$ and $\min\cdot$ were defined in [14] to obtain a detailed classification of optimization problems. Let $\mathcal{C}$ be a class of languages.

$$
\begin{aligned}
f \in \#\cdot\mathcal{C} \quad &\Longleftrightarrow \quad \text{there exists } A \in \mathcal{C} \text{ and } g \in \mathrm{FP} \text{ such that} \\
&\qquad\quad f(x) = \big\|\{\, y \mid y \leq g(x) \wedge (x,y) \in A \,\}\big\|\,. \\
f \in \max\cdot\mathcal{C} \quad &\Longleftrightarrow \quad \text{there exists } A \in \mathcal{C} \text{ and } g \in \mathrm{FP} \text{ such that} \\
&\qquad\quad f(x) = \max\{\, y \mid y \leq g(x) \wedge (x,y) \in A \,\} \\
&\qquad\quad \text{and if this set is empty let } f(x) = 0. \\
f \in \min\cdot\mathcal{C} \quad &\Longleftrightarrow \quad \text{there exists } A \in \mathcal{C} \text{ and } g \in \mathrm{FP} \text{ such that} \\
&\qquad\quad f(x) = \min\{\, y \mid y \leq g(x) \wedge (x,y) \in A \,\} \\
&\qquad\quad \text{and if this set is empty let } f(x) = g(x)+1.
\end{aligned}
$$

Fact 3.1. *Let* $\mathcal{F} \in \{\mathrm{FP}, \mathrm{FP}^{\mathrm{NP}}[O(\log n)], \mathrm{FP}^{\mathrm{NP}}_{\mathrm{tt}}, \mathrm{FP}^{\mathrm{NP}}, \#\cdot\mathrm{P}, \#\cdot\mathrm{NP}, \#\cdot\mathrm{coNP}, \#\cdot\mathrm{P}^{\mathrm{NP}}, \max\cdot\mathrm{P}, \max\cdot\mathrm{NP}, \min\cdot\mathrm{P}, \min\cdot\mathrm{NP}\}$. *Then* $\mathcal{F} = (F, \Phi)$-FP *for* F *and* Φ *as specified in Table 1.*

(F,Φ)-FP	$F(\vec{u})$ with $\vec{u} = (u_1, \ldots, u_k)$	Φ
FP	u_1	Ω
$\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$	u_m with $m = \max\{u_i \mid \log k < i \leq k\}$	$\{\vec{u} \mid \max\{u_i \mid \log k < i \leq k\} \leq \log k\}$
$\mathrm{FP}^{\mathrm{NP}}_{\mathrm{tt}}$	u_{2i-1} if $u_{2i} = \max u_{2j}$	$\{\vec{u} \mid (u_{2i} = u_{2l} = \max u_{2j} \Rightarrow u_{2i-1} = u_{2l-1}) \wedge \max u_{2j} \leq \log k\}$
$\mathrm{FP}^{\mathrm{NP}}$	u_{2i-1} if $u_{2i} = \max u_{2j}$	$\{\vec{u} \mid u_{2i} = u_{2l} = \max u_{2j} \Rightarrow u_{2i-1} = u_{2l-1}\}$
$\#\cdot\mathrm{P}$	$\sum u_j$	Ω
$\#\cdot\mathrm{NP}$	$\mathrm{span}_+(\vec{u})$	Ω
$\#\cdot\mathrm{coNP}$	$u_1 - \mathrm{span}_+(\vec{u})$	$\{\, \vec{u} \mid u_1 = \max u_j \,\}$
$\#\cdot\mathrm{P}^{\mathrm{NP}}$	$\lVert\{z \mid z > 0 \ \wedge\ z \equiv 1 \pmod 2 \ \wedge\ (\exists j)[u_j = z] \ \wedge\ (\forall j)[u_j \neq z+1]\}\rVert$	Ω
$\max\cdot\mathrm{P}$	$\begin{cases} \max\{j \mid u_j > 0\} & \text{if } (\exists j)[u_j > 0] \\ 0 & \text{otherwise} \end{cases}$	Ω
$\max\cdot\mathrm{NP}$	$\max u_j$	Ω
$\min\cdot\mathrm{P}$	$\begin{cases} \min\{j \mid u_j > 0\} & \text{if } (\exists j)[u_j > 0] \\ k+1 & \text{otherwise} \end{cases}$	Ω
$\min\cdot\mathrm{NP}$	$\min\{u_j \mid u_j > 0\} - 1$	$\{\, \vec{u} \mid (\exists j)[u_j > 0] \,\}$

Table 1: Characterizations of some function classes where F maps to $\mathbb{N}$.

Proof. We start with the easy to verify table entries. The characterization of FP is trivial and we already argued for $\#\cdot\mathrm{P}$ in case of Φ being the set of all finite vectors over $\{0,1\}$. If we let $\Phi = \Omega$ we have to say additionally how $\#\cdot\mathrm{P}$ captures the class defined by the generator: For $g, h \in \mathrm{FP}$ we let a witnessing nondeterministic polynomial-time Turing machine M on path i with $i \leq h(x)$ compute a binary tree having exactly $g(x,i)$ accepting paths.

The characterizations of $\max\cdot\mathrm{P}$ and $\min\cdot\mathrm{P}$ correspond in a straightforward way to the original definitions as can be seen as follows. If we simply output on accepting paths 1 and on rejecting paths 0 we capture max$\cdot$ P and min$\cdot$ P with the listed generators. On the other hand, for $g, h \in \mathrm{FP}$ the set A of all pairs (x, y) with $y \leq h(x)$ such that $g(x, y) > 0$ clearly is in P.

$\#\cdot$ NP: Having in mind that $\#\cdot \mathrm{NP} = \text{span-P}$ we easily obtain that the class characterized by the generator with $F = \mathrm{span}_+$ is a subclass of $\#\cdot$ NP. Conversely, let $f \in \#\cdot \mathrm{NP}$ by a set $A \in \mathrm{NP}$ and a function $h \in \mathrm{FP}$ such that $f(x) = \left\|\left\{ y \mid y \leq h(x) \wedge (x, y) \in A \right\}\right\|$. First we guess some y with $y \leq h(x)$, then we simulate one guessed path of the balanced nondeterministic polynomial-time Turing machine for A on input (x, y) and output $y+1$ if the simulation of that path ends accepting, and 0 if not. The sequence of outputs of the whole computation can be described by FP functions g and h' as the sequence $S_x = (g(x, 0), g(x, 1), \ldots, g(x, h'(x)))$, and clearly, $f(x) = \mathrm{span}_+ S_x$. This completes the characterization with the neutral element 0.

max$\cdot$ NP: '$\Rightarrow$' Let $f \in \max\cdot \mathrm{NP}$ via a function $h \in \mathrm{FP}$ and a set $A \in \mathrm{NP}$. If we use the FP functions g and h' from the description above for $\#\cdot$ NP, we can define $g'(x, i) = \max\{0, g(x, i) - 1\}$ and thus, we obtain $f(x) = \max(g'(x, 0), g'(x, 1), \ldots, g'(x, h'(x)))$. '$\Leftarrow$' Let $F = \max u_j$ and $f \in (F, \Omega)$-FP via $g, h \in \mathrm{FP}$. Note that for all x and all $i \leq h(x)$, $g(x, i)$ can be bounded by a suitable FP function in x. Hence the set $\left\{ (x, y) \mid (\exists i)[i \leq h(x) \wedge g(x, i) = y] \right\}$ is in NP and witnesses that $f \in \max\cdot \mathrm{NP}$.

min$\cdot$ NP: If we output $h(x)+1$ instead of 0 in the simulation for $\#\cdot$ NP described above this proves $\min\cdot \mathrm{NP} \subseteq (F, \Phi)$-FP with $F = \min\left\{ u_j \mid u_j > 0 \right\} - 1$ and $\Phi = \left\{ \vec{u} \mid (\exists j)[u_j > 0] \right\}$. This generator allows to append neutral zeroes to the output sequence and ensures that a sequence consisting only of neutral elements does not appear. The converse inclusion is easily seen.

$\#\cdot$ coNP: We can understand $\#\cdot$ coNP as the set of functions f that count the number of lacking outputs between 0 and the largest value that due to the input and the polynomial time constraints can be possibly output by a transducer. To reflect this we give here a characterization of this class with $\Phi \neq \Omega$ which enables us to know *where* in the sequence of outputs we can find the maximal possible output value. To be more precise: '$\Rightarrow$' Let $f \in \#\cdot \mathrm{coNP}$ via $A \in \mathrm{coNP}$ and $h \in \mathrm{FP}$. Again we use the functions g and h' as described above for $\#\cdot$ NP with the difference that we simulate here the machine for $\overline{A} \in \mathrm{NP}$. Now define $g'(x, 0) = h(x) + 2$ and $g'(x, i+1) = g(x, i)$. Then

$$\begin{aligned}
f(x) &= \left\|\left\{ y \mid y \leq h(x) \wedge (x, y) \in A \right\}\right\| \\
&= h(x) + 1 - \left\|\left\{ y \mid y \leq h(x) \wedge (x, y) \in \overline{A} \right\}\right\| \\
&= h(x) + 1 - \left\|\left\{ y+1 \mid y \leq h(x) \wedge (x, y) \in \overline{A} \right\}\right\| \\
&= h(x) + 1 - \mathrm{span}_+\big(g'(x, 1), g'(x, 2), \ldots, g'(x, h'(x)+1)\big) \\
&= h(x) + 2 - \mathrm{span}_+\big(h(x)+2, g'(x, 1), g'(x, 2), \ldots, g'(x, h'(x)+1)\big) \\
&= g'(x, 0) - \mathrm{span}_+\big(g'(x, 0), g'(x, 1), g'(x, 2), \ldots, g'(x, h'(x)+1)\big) \\
&= F(g'(x, 0), g'(x, 1), \ldots, g'(x, h'(x)+1)).
\end{aligned}$$

Moreover, it holds that the sequence S_x produced in this way by g' and $h'+1$ on x always belongs to the desired Φ. '$\Leftarrow$' Let $f \in (F, \Phi)$-FP with $F(\vec{u}) = u_1 - \mathrm{span}_+(u_1, \ldots, u_k)$ and $\Phi = \left\{ \vec{u} \mid u_1 = \max u_j \right\}$ via $g, h \in \mathrm{FP}$. If we define $h'(x) = g(x, 0)$ and $\overline{A} = \left\{ (x, y) \mid y \leq h'(x) \wedge (\exists j)[g(x, j) = y] \right\}$, then $h' \in \mathrm{FP}$ and $A \in \mathrm{coNP}$ together witness that $f \in \#\cdot \mathrm{coNP}$.

$\#\cdot\mathrm{P}^{\mathrm{NP}}$: We are faced with the problem of counting accepting paths of a nondeterministic polynomial-time Turing machine M which is equipped with an NP oracle B. First consider the set of all paths z resulting from the direct simulation of the NP machine for B within the computation tree of M instead of asking any query to the oracle. We continue on every simulated path with the answer provided there. Moreover we may assume that z is odd. For each z and every query on path z we need to verify the simulation's negative answers. To do so we output z on path z and append the appropriate coNP computation while mapping rejecting paths to 0 and accepting paths to $z+1$. The reverse inclusion is easy to see with a suitable NP oracle.

$\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$: '$\Rightarrow$' Let $f \in \mathrm{FP}^{\mathrm{NP}}[O(\log n)]$ via the deterministic polynomial-time oracle Turing machine M and let s be the number of queries to the oracle for some input x. Consider the following nondeterministic polynomial-time Turing machine M'. On input x it does nondeterministically two things in parallel. On the first branch M' guesses i, $1 \leq i \leq 2^s$, and simulates the work of M on input x taking i as the sequence of oracle answers via suitable encoding. On the other branch M' guesses i, $1 \leq i \leq 2^s$, and tries to verify all encoded positive oracle answers by guessing a path of a nondeterministic polynomial-time Turing machine (with the oracle query as input) accepting the NP oracle. Finally M' outputs i if all these verifications were successful. Note that the largest i output in the second part of the machine we just described is the correct oracle answer string and determines what path of the first part of the same machine outputs $f(x)$. The sequence of outputs over all paths of M' can be written as $(g(x,0), g(x,1), \ldots, g(x,h(x))$ for suitable FP functions g and h. It is now not hard to see that $f \in (F,\Phi)$-FP. '$\Leftarrow$' We can find the maximal value in the respective part of the output sequence by a binary search with a logarithmically bounded number of queries to an NP-oracle.

$\mathrm{FP}^{\mathrm{NP}}_{\mathrm{tt}}$: '$\Rightarrow$' We use a characterization from [19]. It states that any function $f \in \mathrm{FP}^{\mathrm{NP}}_{\mathrm{tt}}$ can be computed by a deterministic computation with a logarithmically bounded number of queries to an NP oracle, which is then followed by a nondeterministic computation, where the outputs on all paths are the same (a similar theorem was proved in [30]). We now use a similar approach as above, where we output possible oracle answer strings on even numbered paths, and odd numbered paths are used to compute the function value based on the possible oracle answers. '$\Leftarrow$' We form a list of oracle queries and ask whether there is an output $y = u_{2i}$ for $y \leq \log k$ in $\vec{u}$. This will enable us to find the maximal u_{2i}. Furthermore we ask for any such y bitwise for the output value on path $2i-1$ if y appears on some path $2i$. Note that all maximal u_{2i} have the same u_{2i-1} and that the length of each u_{2i-1} is bounded by $\log k$.

$\mathrm{FP}^{\mathrm{NP}}$: The characterization of $\mathrm{FP}^{\mathrm{NP}}$ can be deduced from the proof of the characterization of $\mathrm{FP}^{\mathrm{NP}}_{\mathrm{tt}}$ by only dropping the logarithmic bound for the number of oracle queries needed as well as for the values of the sequences. ❑

It is interesting to see that since $\#\cdot\mathrm{coNP}$ is known to be equal to $\#\cdot\mathrm{P}^{\mathrm{NP}}$ [20, 21], we have two substantially different characterizations of the same class, one with $\Phi = \Omega$ while for the other one $\Phi \subsetneq \Omega$.

Next we want to show how restricting the generator set Φ can be used to capture promises about the computation model. We recall the definition of the operator $\#_{\mathrm{few}}\cdot$ from [13]. A function f is in $\#_{\text{few}}\cdot\mathcal{C}$ iff $f \in \#\cdot\mathcal{C}$ and there exists a polynomial p such that for all x it holds that $f(x) \le p(|x|)$. Thus, in case of $\#_{\text{few}}\cdot \mathrm{P}$ we have to deal with at most a polynomial number of accepting paths.

As a second example, we turn to cluster sets A of pairs (x, y) such that for every fixed x the words in the set $A_x = \{\, y \mid (x, y) \in A \,\}$, interpreted as natural numbers, form an integer interval. The class $\mathrm{c}\#\cdot\mathcal{C}$ consists of those functions $f \in \#\cdot\mathcal{C}$ such that the witnessing set $A \in \mathcal{C}$ is a cluster set. In case of $\mathrm{c}\#\cdot\mathrm{P}$ this means that functions are defined by Turing machines where all accepting paths are neighbored. One can view this as a generalization of UP computations and it is known that $\mathrm{c}\#\cdot\mathrm{P} = \#\cdot\mathrm{P}$ if and only if $\mathrm{PP} = \mathrm{UP}$. For a more detailed discussion, see [22].

Note that the generator functions in Table 2 are in all cases the same as in the characterization of the non-promise version of the respective class. Here the generator set makes the difference. We write Ω_1 for Ω_s with $s(n) =_{\text{def}} 1$ for all n.

Fact 3.2. *Let $\mathcal{F} \in \{\#_{\text{few}}\cdot\mathrm{P}, \mathrm{c}\#\cdot\mathrm{P}, \mathrm{c}\#\cdot\mathrm{NP}, \mathrm{c}\#\cdot\mathrm{coNP}\}$. Then $\mathcal{F} = (F, \Phi)$-FP for F and Φ as specified in Table 2.*

(F, Φ)-FP	$F(\vec{u})$ with $\vec{u} = (u_1, \dots, u_k)$	Φ
$\#_{\text{few}}\cdot\mathrm{P}$	$\sum u_j$	$\{\, \vec{u} \in \Omega_1 \mid \|\{j \mid u_j > 0\}\| \le \log k \,\}$
$\mathrm{c}\#\cdot\mathrm{P}$	$\sum u_j$	$\{\vec{u} \in \Omega_1 \mid (\exists \ell, r)\,[\{j \mid u_j > 0\} = [\ell, r]]\,\}$
$\mathrm{c}\#\cdot\mathrm{NP}$	$\mathrm{span}_+(\vec{u})$	$\{\, \vec{u} \mid (\exists \ell, r)\,[\{u_1, \dots, u_k\} \setminus \{0\} = [\ell, r]] \,\}$
$\mathrm{c}\#\cdot\mathrm{coNP}$	$u_1 - \mathrm{span}_+(\vec{u})$	$\{\, \vec{u} \mid u_1 = \max u_j \,\} \cap$ $\{\, \vec{u} \mid (\exists \ell, r)\,[[1, \max u_j] \setminus \{u_1, \dots, u_k\} = [\ell, r]] \,\}$

Table 2: Characterizations of some function classes where F maps to $\mathbb{N}$ and $\Phi \subsetneq \Omega$.

The class GapP of functions f that provide the number of accepting paths minus the number of rejecting paths of nondeterministic (possibly unbalanced) nondeterministic polynomial-time Turing machines was introduced and studied in [10]. Above we already gave a characterization in terms of a generator (F, Φ) with $F = \sum u_j$ and Φ being the set of all finite vectors over $\{-1, 1\}$. To provide a second generator characterization of GapP here, this time with $\Phi = \Omega$, we use the equation $\mathrm{GapP} = \#\cdot\mathrm{P} - \#\cdot\mathrm{P}$ and its relativized version $\mathrm{GapP}^{\mathrm{NP}} = \#\cdot\mathrm{P}^{\mathrm{NP}} - \#\cdot\mathrm{P}^{\mathrm{NP}}$ ([10], see also [13]).

Fact 3.3. *Let $\mathcal{F} \in \{\mathrm{GapP}, \mathrm{GapP}^{\mathrm{NP}}\}$. Then $\mathcal{F} = (F, \Phi)$-FP for F and $\Phi = \Omega$ as specified in Table 3.2.*

Finally, we turn to classes of partial *multi-valued* functions, i.e., partial functions f that map to $2^{\mathbb{N}}$. The following definitions are taken from [31] (see also [3]). From now on let M be

(F,Φ)-FP	$F(\vec{u})$ with $\vec{u} = (u_1, \ldots, u_k)$
GapP	$\\|\{\, j \mid u_j > 1 \,\}\\| \;-\; \\|\{\, j \mid u_j = 1 \,\}\\|$
$\mathrm{GapP}^{\mathrm{NP}}$	$\\|\{\, z \mid z > 0 \;\wedge\; z \equiv 1(2) \;\wedge\; (\exists j)[u_j = z] \,\}\\| \;-$ $\\|\{\, z \mid z > 0 \;\wedge\; z \equiv 0(2) \;\wedge\; (\exists j)[u_j = z] \,\}\\|$

Table 3: Characterizations of some function classes where F maps to $\mathbb{Z}$ and $\Phi = \Omega$.

a balanced nondeterministic polynomial-time Turing transducer. Note that we can make that assumption without loss of generality. Let $M(x)$ denote the set of outputs made on accepting paths of M on input x.

$$f \in \mathrm{NPMV} \iff \text{there exists a nondeterministic polynomial-time Turing transducer } M \text{ such that } f(x) = \begin{cases} M(x) & \text{if } M(x) \neq \emptyset, \\ \perp & \text{otherwise.} \end{cases}$$

A function $f \in \mathrm{NPMV}$ belongs to FewFP iff the witnessing transducer M additionally fulfills the promise that there is some polynomial p such that $\|M(x)\| \leq p(|x|)$ for all x. If this restriction is strengthened to $\|M(x)\| \leq 1$ we get the class NPSV of so-called partial *single-valued* functions. Additionally we define here the class NPUV which is a non-promise version of NPSV and was studied in [30]. This class of *unique-valued* functions can be obtained if we omit the promise $\|M(x)\| \leq 1$ in the definition of NPSV and set the function undefined if $\|M(x)\| > 1$. This reflects the fact that the different computation paths of M did not agree on a single function value for f. It is known that NPUV = NPSV if and only if NP = coNP.

$$f \in \mathrm{NPUV} \iff \text{there exists a nondeterministic polynomial-time Turing transducer } M \text{ such that } f(x) = \begin{cases} M(x) & \text{if } \|M(x)\| = 1, \\ \perp & \text{otherwise.} \end{cases}$$

The characterizations of the mentioned classes in Table 4 are direct translations of the definitions. Note that span_+ is by definition the cardinality of all values of nonzero components of a vector. Again, the promise conditions on the computation model are reflected in a natural way in the respective definitions of Φ.

Fact 3.4. *Let $\mathcal{F} \in \{\mathrm{NPMV}, \mathrm{FewFP}, \mathrm{NPSV}, \mathrm{NPUV}\}$. Then $\mathcal{F} = (F,\Phi)$-FP for F and Φ as specified in Table 4.*

(F,Φ)-FP	$F(\vec{u})$ with $\vec{u}=(u_1,\ldots,u_k)$	Φ
NPMV	$\begin{cases} \{\, u_j - 1 \mid u_j > 0 \,\} & \text{if } (\exists j)[u_j > 0] \\ \bot & \text{otherwise} \end{cases}$	Ω
FewFP	$\begin{cases} \{\, u_j - 1 \mid u_j > 0 \,\} & \text{if } (\exists j)[u_j > 0] \\ \bot & \text{otherwise} \end{cases}$	$\{\, \vec{u} \mid \mathrm{span}_+(\vec{u}) \leq \log k \,\}$
NPSV	$\begin{cases} \{\, u_j - 1 \mid u_j > 0 \,\} & \text{if } (\exists j)[u_j > 0] \\ \bot & \text{otherwise} \end{cases}$	$\{\, \vec{u} \mid \mathrm{span}_+(\vec{u}) \leq 1 \,\}$
NPUV	$\begin{cases} \{\, u_j - 1 \mid u_j > 0 \,\} & \text{if } \mathrm{span}_+(\vec{u}) = 1 \\ \bot, & \text{otherwise} \end{cases}$	Ω

Table 4: Characterizations of some function classes where F maps to $2^{\mathbb{N}}$.

3.3 Restricting the Set of Admissible Sequences

We conclude this section with a technical lemma. It shows that we can restrict ourselves to generator sets with elements that satisfy a certain relationship between the number of components and their values in the polynomial-time setting. Recall the definition of Ω_s where the values of all elements of a vector are less-than or equal to the value of function s depending on the dimension of the vector.

Lemma 3.5. *Let (F,Φ) be a paddable generator, and let s be a total function satisfying $s \geq_{a.e.} id$. Then (F,Φ)-FP $= (F,\Phi\cap\Omega_s)$-FP.*

Proof. We only have to prove the inclusion from left to right, so suppose $f \in (F,\Phi)$-FP and let k_0 be minimal such that $s(n) \geq n$ for all $n \geq k_0$. Then there are functions $g,h \in \mathrm{FP}$ such that $\big(g(x,0),g(x,1),\ldots,g(x,h(x))\big) \in \Phi$ and $f(x) \simeq F\big(g(x,0),g(x,1),\ldots,g(x,h(x))\big)$ for all x. Moreover, there exist polynomials p_1,p_2 with $g(x,j) < 2^{p_1(|x|+|j|)}$ and $h(x) < 2^{p_2(|x|)}$ for all x. Since (F,Φ) is paddable, there is a neutral element γ such that $F(\vec{u}) \simeq F(\vec{u},\gamma) \simeq F(\vec{u},\gamma,\gamma) \simeq \ldots$, and $\vec{u}\in\Phi \iff (\vec{u},\gamma)\in\Phi \iff (\vec{u},\gamma,\gamma)\in\Phi \iff \ldots$ for all $\vec{u}\in\Omega$.

Case 1. Suppose $\Phi\cap\Omega_s = \emptyset$. Then Φ has to be empty due to the paddability of (F,Φ). Hence, (F,Φ)-FP was empty as well.

Case 2. Now, $\Phi\cap\Omega_s \neq \emptyset$. Define functions g' and h' as

$$g'(x,j) = \begin{cases} g(x,j) & \text{if } j \leq h(x), \\ \gamma & \text{otherwise,} \end{cases}$$

$$h'(x) = \max\left\{\gamma, 2^{p_1(|x|+p_2(|x|))}, 2^{p_2(|x|)}, k_0\right\}.$$

Clearly, $g', h' \in \mathrm{FP}$, and for every $x \in \Sigma^*$ we have $\big(g'(x,0), g'(x,1), \ldots, g'(x,h'(x))\big) \in \Phi \cap \Omega_s$ and $f(x) \simeq F\big(g(x,0), g(x,1), \ldots, g(x,h(x))\big) \simeq F\big(g'(x,0), g'(x,1), \ldots, g'(x,h'(x))\big)$. Hence, $f \in (F, \Phi \cap \Omega_s)$-FP. ❑

4 Reducibilities

It will be our aim in the upcoming sections to compare function classes with respect to their complexity. For this end we will establish certain relations between the defining generators. This is made precise formally by defining suitable *reducibilities* between generators.

Definition 4.1. Let (F, Φ) and (G, Γ) be generators. Then we say that (F, Φ) is *polylogarithmic-time component-wise many-one reducible* (FPLT reducible, for short) to (G, Γ), in symbols $(F, \Phi) \leq_m^{\mathrm{FPLT}} (G, \Gamma)$, if and only if there exist functions $d, e \in \mathrm{FPLT}$ such that $S_{\vec{u}} =_{\mathrm{def}} \big(d(\vec{u}, 0), d(\vec{u}, 1), \ldots, d(\vec{u}, e(\vec{u}))\big) \in \Gamma$ and $F(\vec{u}) \simeq G\big(S_{\vec{u}}\big)$ for every $\vec{u} \in \Phi$.

Observe that in the applications, since (F, Φ) is a generator, this means that the argument $\vec{u}$ of F is an output sequence of an nondeterministic polynomial-time Turing machine M, hence the length of $\vec{u}$ might still be exponential in the length n of the input of M. Therefore, the polylogarithmic time above translates to "time polynomial in n." Also, it is worth to note that the length of $S_{\vec{u}}$ might still be exponential in n.

The following proposition is our main motivation for FPLT reductions.

Proposition 4.2. *If* $(F, \Phi) \leq_m^{\mathrm{FPLT}} (G, \Gamma)$*, then* (F, Φ)-FP $\subseteq (G, \Gamma)$-FP*. Moreover, this statement relativizes to any oracle function.*

Proof. Suppose $f \in (F, \Phi)$-FP. Because of Lemma 3.5, there are functions $g, h \in \mathrm{FP}$ such that $\big(g(x,0), g(x,1), \ldots, g(x,h(x))\big) \in \Phi \cap \Omega_{id}$ and $f(x) \simeq F\big(g(x,0), g(x,1), \ldots, g(x,h(x))\big)$ for every $x \in \Sigma^*$. By assumption, $(F, \Phi \cap \Omega_{id}) \leq_m^{\mathrm{FPLT}} (G, \Gamma)$ via $d, e \in \mathrm{FPLT}$. Defining $g'(x,j) = d\big(g(x,0), g(x,1), \ldots, g(x,h(x)), j\big)$ and $h'(x) = e\big(g(x,0), g(x,1), \ldots, g(x,h(x))\big)$ gives $\big(g'(x,0), g'(x,1), \ldots, g'(x,h'(x))\big) \in \Gamma$ and

$$f(x) \simeq F\big(g(x,0), g(x,1), \ldots, g(x,h(x))\big) \simeq G\big(g'(x,0), g'(x,1), \ldots, g'(x,h'(x))\big).$$

It only remains to argue that $g', h' \in \mathrm{FP}$. This is easily seen if we compute only those parts of the sequence $\big(g(x,0), g(x,1), \ldots, g(x,h(x))\big)$ that are actually needed by e and d. Then, in both cases, the run-time can be bounded by $c \log^k 2^n = cn^k$ for appropriate $c, k \in \mathbb{N}$, where n is the length of the input, hence polynomial in n. This proof relativizes. ❑

Thus we have a tool that applies *en-masse* to all the function classes characterized in Sect. 3 and more. However, Proposition 4.2 seems not to yield an equivalence to the containment of the function classes, i.e., we have no reverse implication. Our main technical result, proved in the next section, is that we do get an equivalence—to the *relativized containment* of the function classes—by substituting a slightly weaker reducibility for $\leq_m^{\mathrm{FPLT}}$, defined next:

Definition 4.3. Let (F,Φ) and (G,Γ) be generators and let s be a total function from $\mathbb{N} \to \mathbb{N}$. We say that (F,Φ) is *s-partially polylogarithmic time component-wise many-one reducible* (s-partially FPLT reducible, for short) to (G,Γ), in symbols $(F,\Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$, if and only if $(F,\Phi \cap \Omega_s) \leq_m^{\mathrm{FPLT}} (G,\Gamma)$.

Next we give some properties of these reducibilities. The following relations between total and partial reducibility and between different forms of partial reducibilities are obvious:

Proposition 4.4. *Let (F,Φ) and (G,Γ) be generators.*

1. *If $(F,\Phi) \leq_m^{\mathrm{FPLT}} (G,\Gamma)$ then $(F,\Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$ for every total function s.*
2. *Let s,t be total functions such that $s \leq t$. If $(F,\Phi) \leq_{t\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$ then $(F,\Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$.*

The motivation for the introduction of partial FPLT reductions is, that they allow an equivalence between reducibility of generators and relativizing containment of classes. Certainly, the question of whether these reductions are strictly weaker than (total) FPLT reductions arises. Next, we show that the answer is yes and that the implications given in Proposition 4.4 are not equivalences by constructing suitable counterexamples.

Proposition 4.5 proves that the reverse implication of the second statement of Proposition 4.4 is false.

Proposition 4.5. *Let s and t be total functions, let t be strictly larger than s on at least one argument. There are generators (F,Φ) and (G,Γ) such that $(F,\Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$ yet $(F,\Phi) \not\leq_{t\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$.*

Proof. Let s and t be such functions. Define $\Phi = \Omega$ and $\Gamma = \{(1,1)\}$. Define $F(\vec{u}) = 1$ for all $\vec{u} \in \Omega_s$ and $F(\vec{u}) = 2$ for all $\vec{u} \in \Omega \setminus \Omega_s$. Define $G((1,1)) = 1$. It is not hard to see that $(F,\Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$ via the FPLT functions d and e where $e(\vec{u}) = 2$ and $d(\vec{u},i) = 1$ for $i = 1,2$. Since $\Omega_t \setminus \Omega_s \neq \emptyset$, no functions d', e' can $\leq_{t\text{-}m}^{\mathrm{FPLT}}$ reduce (F,Φ) to (G,Γ). ❑

Theorem 4.6 proves that the reverse implication of the first statement of Proposition 4.4 is false.

Theorem 4.6. *There are (recursive) generators (F,Φ) and (G,Γ) with $(F,\Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$ for all total functions s, but $(F,\Phi) \not\leq_m^{\mathrm{FPLT}} (G,\Gamma)$.*

Proof. We construct (F,Φ) and (G,Γ) in stages. For that, we only consider one-dimensional vectors $\vec{u}$, i.e., natural numbers. Let $\{[0],[1],\ldots,[m],\ldots\}$ be an enumeration of all pairs of FPLT functions where $[m]$ means a pair (g,h) of FPLT functions and $[m](n)$, means the sequence $\big(g(n,0), g(n,1), \ldots, g(n,h(n))\big)$.

Stage $m = 0$: Define $\Phi_0 = \{0\}$, $\Gamma_0 = \{[0](0)\}$, and $F_0(0) = G_0\big([0](0)\big) = 0$. Hence, the pair $[0]$ shows $(F_0,\Phi_0) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G_0,\Gamma_0)$ for all s with $s(1) = 0$.

Stage $m \geq 1$: Let j_{m-1} be an index of a pair of FPLT functions with generators (F_{m-1}, Φ_{m-1}) and (G_{m-1}, Γ_{m-1}), and let r_{m-1} be an associated natural number such that $[j_{m-1}]$ witnesses $(F_{m-1}, \Phi_{m-1}) \leq^{\mathrm{FPLT}}_{s\text{-}m} (G_{m-1}, \Gamma_{m-1})$ for every s with $s(1) \leq r_{m-1}$. Define j_m to be the least natural number greater than j_{m-1} which satisfies

(i) $[j_m](n) \in \Gamma_{m-1}$ and $F_{m-1}(n) \simeq G_{m-1}\big([j_m](n)\big)$ for all $n \leq r_{m-1}$,

(ii) there is an $r > r_{m-1}$ so that $[j_{m-1}](r) \neq [j_m](r)$ and, if $[j_m](r) \in \Gamma_{m-1}$, then $G_{m-1}\big([j_m](r)\big) \not\simeq G_{m-1}\big([j_{m-1}](r)\big)$.

The existence of such j_m is assured since FPLT is closed under finite variations. Once j_m is given, let r_m be the smallest r satisfying condition (ii). Now, define for the generator sets $\Phi_m = \Phi_{m-1} \cup \{r_m\}$, $\Gamma_m = \Gamma_{m-1} \cup \big\{[j_m](r_m)\big\}$, and for the generator functions $F_m(n) = F_{m-1}(n)$ for all $n \in \Phi_{m-1}$, and $G_m(n) = G_{m-1}(n)$ for all $n \in \Gamma_{m-1}$. We set $F_m(r_m) = G_m\big([j_m](r_m)\big)$, so it remains to define the latter one. If $[j_m](r_m) \in \Gamma_{m-1}$, then $G_m\big([j_m](r_m)\big)$ is already defined, otherwise define $G_m\big([j_m](r_m)\big) = m$. Hence, we have that $[j_m]$ s-partially reduces (F_m, Φ_m) to (G_m, Γ_m) for all s with $s(1) \leq r_m$, and for no pair of FPLT functions with index $n < j_m$ and no function s with $s(1) \geq r_m$, it holds $(F_m, \Phi_m) \leq^{\mathrm{FPLT}}_{s\text{-}m} (G_m, \Gamma_m)$ via $[n]$.

Since for all $m \in \mathbb{N}$, (F_{m+1}, Φ_{m+1}) is an extension of (F_m, Φ_m), and (G_{m+1}, Γ_{m+1}) is an extension of (G_m, Γ_m), we can define (F, Φ) and (G, Γ) as the limits of the stage construction for $m \to \infty$. By the construction it follows that, for all s, it holds $(F, \Phi) \leq^{\mathrm{FPLT}}_{s\text{-}m} (G, \Gamma)$, since there always exists an m such that $s(1) \leq r_m$. On the other hand, for all pairs of FPLT functions (g, h) there is an $r \in \Phi$ with $\big(g(r,0), \ldots, g(r, h(r))\big) \notin \Gamma$ or $F(r) \not\simeq G\big(g(r,0), \ldots, g(r, h(r))\big)$. Thus, $(F, \Phi) \not\leq^{\mathrm{FPLT}}_{m} (G, \Gamma)$. ❑

Concerning reflexivity of our reductions, the following has to be said: Because of the sub-linear time restriction, the total FPLT reductions are not reflexive (think of vectors consisting of only one component: to compute the identity function here requires clearly linear time as already mentioned in the preliminary section). Also, there are functions s such that s-partial FPLT reductions are not reflexive. For instance, let $s(n) = 2^n$ and consider vectors, where all components except the first are 0. Then the length of the first component can be linear in the length of the vector, hence again the identity cannot be computed in sub-linear time. However, restricted to paddable generators, s-partial FPLT reductions are reflexive for all s (this follows from Theorem 5.1 below, since clearly every class is relativizably contained in itself).

Next we note that both reducibility notions fulfill some form of transitivity.

Theorem 4.7. *1. $\leq^{\mathrm{FPLT}}_{m}$ is transitive.*

2. Let $(F, \Phi), (G, \Gamma)$, and (H, Π) be generators where (G, Γ) is paddable, and let t be a total function with $t \geq_{a.e.} id$. If $(F, \Phi) \leq^{\mathrm{FPLT}}_{s\text{-}m} (G, \Gamma)$ for an arbitrary total function s, and $(G, \Gamma) \leq^{\mathrm{FPLT}}_{t\text{-}m} (H, \Pi)$, then $(F, \Phi) \leq^{\mathrm{FPLT}}_{s\text{-}m} (H, \Pi)$.

Proof. For *1* suppose $(F, \Phi) \leq^{\mathrm{FPLT}}_{m} (G, \Gamma)$ via $d_1, e_1 \in$ FPLT, and $(G, \Gamma) \leq^{\mathrm{FPLT}}_{m} (H, \Pi)$ via $d_2, e_2 \in$ FPLT. We will define functions $d, e \in$ FPLT such that $\big(d(\vec{u}, 0), d(\vec{u}, 1), \ldots, d(\vec{u}, e(\vec{u}))\big) \in$

Π and $F(\vec{u}) \simeq H\big(d(\vec{u},0), d(\vec{u},1), \ldots, d(\vec{u}, e(\vec{u}))\big)$ for all $\vec{u} \in \Phi$. For this, let $e(\vec{u}) = e_2\big(\big(d_1(\vec{u},0), d_1(\vec{u},1), \ldots, d_1(\vec{u}, e_1(\vec{u}))\big)\big)$, and let $d(\vec{u},j) = d_2\big(\big(d_1(\vec{u},0), d_1(\vec{u},1), \ldots, d_1(\vec{u}, e_1(\vec{u}))\big), j\big)$.

We have to show that both functions are polylogarithmic-time computable. Consider a machine M_e for e that on input $\vec{u}$, first computes $c = e_1(\vec{u})$, then starts the computation of e_2 by simulating a polylogarithmic-time machine M_{e_2} computing e_2; if during this simulation M_e needs to access the m-th component of the input vector for M_{e_2}, then M_e behaves as follows: if $m \leq c$ then M_e computes $d_1(\vec{u},m)$, otherwise M_e continues with the simulation of M_{e_2} as in the case of an empty input tape. Clearly M_e computes e. Note that M_e is polylogarithmic time-bounded, and thus $e \in$ FPLT. Analogously, one can define a machine M_d for d that works on input vectors $(\vec{u}, j)$. Hence also $d \in$ FPLT.

2. can be shown via a simultaneous application of the techniques used for proving Lemma 3.5 and the first statement. ❑

Corollary 4.8. *Let $(F,\Phi), (G,\Gamma)$, and (H,Π) be generators. If $(F,\Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$ for an arbitrary total function s and $(G,\Gamma) \leq_{m}^{\mathrm{FPLT}} (H,\Pi)$, then $(F,\Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (H,\Pi)$.*

Corollary 4.9. *Restricted to paddable generators, $\leq_{s\text{-}m}^{\mathrm{FPLT}}$ is transitive for every total function $s \geq_{a.e.} id$ (thus in particular, $\leq_{id\text{-}m}^{\mathrm{FPLT}}$ is transitive).*

5 Comparing Function Classes

In this section, we state and prove our main technical result. Proposition 4.2 in the previous section motivates FPLT reductions between generators by relating them to relativizing containment of generated function classes. Here, we establish an equivalence between partial FPLT reductions and relativizing inclusions of function classes. The crucial point why such an equivalence works is a quantifier swapping of the following kind: "For every oracle there is a machine ..." to "there is a machine such that for every oracle ...."

As mentioned in the introductory section, Theorem 5.1 proves once and for all the diagonalization part of every separating oracle construction between two function classes characterized by generators.

Theorem 5.1. *Let (F,Φ) be a paddable generator, and let (G,Γ) be a generator. Then the following statements are equivalent:*

1. *$(F,\Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$ for every total function s.*

2. *$(F,\Phi) \leq_{id\text{-}m}^{\mathrm{FPLT}} (G,\Gamma)$.*

3. *(F,Φ)-$\mathrm{FP}^{\alpha} \subseteq (G,\Gamma)$-$\mathrm{FP}^{\alpha}$ for every total polynomially length-bounded function α.*

Proof. *1* $\Longrightarrow$ *2*: Trivial.

2 $\Longrightarrow$ *3*: Consequence of Lemma 3.5 and Proposition 4.2.

3 $\Longrightarrow$ *1*: First we introduce some notations. Define the function σ as

$$\sigma(n) = \sum_{j=0}^{n-1}(j+1) = n + \frac{n(n-1)}{2}.$$

Clearly, $\sigma \in \mathrm{FP}$, and, given a natural number n, there is exactly one $j \geq 0$ such that $\sigma(j) \leq n \leq \sigma(j) + j$. In the proof below we use σ to define functions in stages, one interval $[\sigma(n), \sigma(n)+n]$ per stage, and since the n values will be different for different stages there will be no collisions.

For a given set $\mathcal{C} \subseteq \Omega$ with $d_{\mathcal{C}} =_{\mathrm{def}} \min_{\vec{u}\in\mathcal{C}}(\dim \vec{u} - 1)$, define $A(\mathcal{C})$ to be the following class of total functions:

$$A(\mathcal{C}) = \left\{ \alpha \;\middle|\; (\forall n, n \geq d_{\mathcal{C}}) \left[\big(\alpha(\sigma(n)), \alpha(\sigma(n)+1), \ldots, \alpha(\sigma(n)+n)\big) \in \mathcal{C} \right] \right\}.$$

Later $\mathcal{C}$ will be a generator set, and $A(\mathcal{C})$ will be the set of *admissible* functions, i.e., functions for which, during the application we have in mind, only input vectors from $\mathcal{C}$ are produced.

Note that for a given generator set $\mathcal{C}$ with neutral element γ (i.e., $\vec{u} \in \mathcal{C}$ if and only if $(\vec{u}, \gamma) \in \mathcal{C}$) and for every given vector $\vec{w} = (w_1, \ldots, w_k) \in \mathcal{C}$ the function $\alpha_{\vec{w}}$ defined as

$$\alpha_{\vec{w}}(x) = \begin{cases} w_{\mathrm{val}(x)-l(j)} & \text{if } l(j) \leq \mathrm{val}(x) \leq l(j)+k-1 \text{ for some } j \geq 0, \\ \gamma & \text{otherwise,} \end{cases}$$

where $l(j)$ denotes $\sigma(k-1+j)$, belongs to $A(\mathcal{C})$. This can be seen from the observation that the sequence of values of $\alpha_{\vec{w}}$ has the form $\gamma, \gamma, \ldots, \gamma, w_1, w_2, \ldots, w_k, w_1, w_2, \ldots, w_k, \gamma, w_1, w_2, \ldots, w_k, \gamma, \gamma, w_1, w_2, \ldots$.

We now start with the proof. Let (F, Φ) and (G, Γ) be generators where (F, Φ) is paddable. Suppose it that $(F,\Phi)\text{-}\mathrm{FP}^{\alpha} \subseteq (G,\Gamma)\text{-}\mathrm{FP}^{\alpha}$ holds for all total, polynomially length-bounded functions α. We have to show that $(F, \Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G, \Gamma)$ for all total functions s. By Proposition 4.4 we may assume without loss of generality that for all n, $s(n) \geq \mathrm{val}(\gamma)$, where γ is the neutral element of (F, Φ). We are going to prove that there are functions $d, e \in \mathrm{FPLT}$ such that $\big(d(\vec{w}, 0), d(\vec{w}, 1), \ldots, d(\vec{w}, e(\vec{w}))\big) \in \Gamma$ and $F(\vec{w}) \simeq G\big(d(\vec{w}, 0), d(\vec{w}, 1), \ldots, d(\vec{w}, e(\vec{w}))\big)$ for every $\vec{w} \in \Phi \cap \Omega_s$. Define a *test function* τ_α for an arbitrary (oracle) function $\alpha \in A(\Phi)$ as

$$\tau_\alpha(x) \simeq F\Big(\alpha\big(\sigma(\max(x, d_\Phi))\big), \alpha\big(\sigma(\max(x, d_\Phi)) + 1\big), \ldots, \alpha\big(\sigma(\max(x, d_\Phi)) + \max(x, d_\Phi)\big)\Big),$$

where for simplicity we identify x with $\mathrm{val}(x)$.

Certainly $\tau_\alpha \in (F,\Phi)\text{-}\mathrm{FP}^{\alpha}$ for every polynomially length-bounded $\alpha \in A(\Phi)$ (this is even witnessed by the same pair of transducers independent of the oracle α), and thus by our assumption we get $\tau_\alpha \in (G,\Gamma)\text{-}\mathrm{FP}^{\alpha}$ for every $\alpha \in A(\Phi)$. Consider an effective enumeration of all pairs of deterministic oracle Turing transducers such that the run-time of both machines of the k-th pair is bounded by $n^k + k$ ($k \in \mathbb{N}$). Denote the pair of functions (g^α, h^α) computed by the k-th pair of machines from the enumeration, both using oracle function α, by $[k]^\alpha$. That is, $[k]^\alpha$ maps an $x \in \Sigma^*$ to the sequence $\big(g^\alpha(x, 0), g^\alpha(x, 1), \ldots, g^\alpha(x, h^\alpha(x))\big)$. Since $\tau_\alpha \in (G,\Gamma)\text{-}\mathrm{FP}^{\alpha}$ for every $\alpha \in A(\Phi)$ there is a $k \in \mathbb{N}$ (depending on α) such that for all $x \in \Sigma^*$,

$$[k]^\alpha(x) \in \Gamma \text{ and } \tau_\alpha(x) \simeq G([k]^\alpha(x)). \tag{3}$$

We now claim that k may even be chosen independently of α:

Claim. *There exists an index $k_0 \in \mathbb{N}$ such that for all polynomially length-bounded functions $\alpha \in A(\Phi \cap \Omega_s)$ and all $x \in \Sigma^*$,*

$$[k_0]^\alpha(x) \in \Gamma \text{ and } \tau_\alpha(x) \simeq G([k_0]^\alpha(x)).$$

Assume for a moment the claim holds, and let $[k_0]^{(\cdot)} = (g^{(\cdot)}, h^{(\cdot)})$. Define the functions d, e as

$$\begin{aligned} d(\vec{w}, j) &= g^{\alpha_{\vec{w}}}(\dim \vec{w} - 1, j), \\ e(\vec{w}) &= h^{\alpha_{\vec{w}}}(\dim \vec{w} - 1). \end{aligned}$$

Then $d, e \in \mathrm{FPLT}$ as can be seen by considering the following procedure: Simulate the computation of g and h respectively, and during this simulation replace oracle access by access to the input vector (computing the needed indices can be done in polylogarithmic time).

Now, let $\vec{w} \in \Phi \cap \Omega_s$. Since Φ is paddable and Ω_s is paddable with respect to γ (note that $\gamma \leq s(n)$ for all n) we have that $\Phi \cap \Omega_s$ is paddable and thus we have $\alpha_{\vec{w}} \in A(\Phi \cap \Omega_s)$ as noted above. From the claim we immediately see that $\big(d(\vec{w}, 0), d(\vec{w}, 1), \ldots, d(\vec{w}, e(\vec{w}))\big) \in \Gamma$ and

$$F(\vec{w}) \simeq \tau_{\alpha_{\vec{w}}}(\dim \vec{w} - 1) \simeq G([k_0]^{\alpha_{\vec{w}}}(\dim \vec{w} - 1)) \simeq G\big(d(\vec{w}, 0), d(\vec{w}, 1), \ldots, d(\vec{w}, e(\vec{w}))\big).$$

Hence, $(F, \Phi) \leq_{s\text{-}m}^{\mathrm{FPLT}} (G, \Gamma)$.

Proof of claim. The proof is by contradiction, i.e., we assume that for every $k \in \mathbb{N}$ there is a polynomially length-bounded function $\alpha \in A(\Phi \cap \Omega_s)$ and an input $x \in \Sigma^*$ such that

$$[k]^\alpha(x) \notin \Gamma \text{ or } \tau_\alpha(x) \not\simeq G([k]^\alpha(x)). \tag{4}$$

In order to get a contradiction to (4) we construct a sequence of partial functions $\{\beta_m\}_{m \in \mathbb{N}}$ and a sequence of words $\{x_m\}_{m \in \mathbb{N}}$ satisfying for all $m \geq 0$ the following conditions:

(a) $\beta_{m-1}(x) = \beta_m(x)$ for every $x \in D_{\beta_{m-1}}$,

(b) $D_{\beta_m} = [0, \sigma(n_m) + n_m]$ for a suitable n_m such that during the whole computation of $[m]^{\beta_m}(x_m)$ all questions to β_m are contained in D_{β_m},

(c) for all $x \leq n_m$, $(\beta_m(\sigma(x)), \beta_m(\sigma(x) + 1), \ldots, \beta_m(\sigma(x + 1))) \in \Phi \cap \Omega_s$, in other words, β_m restricted to its domain behaves like a function from $A(\Phi \cap \Omega_s)$,

(d) $[m]^{\beta_m}(x_m) \notin \Gamma$ or $\tau_{\beta_m}(x_m) \not\simeq G([m]^{\beta_m}(x_m))$.

Using these sequences define the total function $\alpha_0 = \lim_{m \to \infty} \beta_m$, i.e., $\alpha_0(x) = \beta_m(x)$ for an $m \in \mathbb{N}$ with $x \in D_{\beta_m}$. Hence, $\alpha_0 \in A(\Phi \cap \Omega_s)$ and for all $m \in \mathbb{N}$ there is an $x_m \in \Sigma^*$ such that $[m]^{\alpha_0}(x_m) \notin \Gamma$ or $\tau_{\alpha_0}(x_m) \not\simeq G([m]^{\alpha_0}(x_m))$. This is a contradiction to (3), and thus our claim holds.

We construct the needed sequences by a stage construction.

Stage $m = -1$. Set $D_{\beta_{-1}} = \emptyset$ for initialization.

Stage $m \geq 0$. Consider the following assumption: For every polynomially length-bounded function $\alpha \in A(\Phi \cap \Omega_s)$ with $\alpha|_{D_{\beta_{m-1}}} \simeq \beta_{m-1}$ and for every $x \in \Sigma^*$ we have

$$[m]^\alpha(x) \in \Gamma \text{ and } \tau_\alpha(x) \simeq G([m]^\alpha(x)). \tag{5}$$

If this assumption is false, then the falsifying witnesses α and x can be used to define $\beta_m = \alpha|_{[0,\sigma(n_m)+n_m]}$ by choosing n_m large enough to guarantee the conditions (a) and (b) with respect to x. The other conditions (choosing $x_m = x$) follow from $\alpha \in A(\Phi \cap \Omega_s)$ with $\alpha|_{D_{\beta_{m-1}}} \simeq \beta_{m-1}$ and since α and x contradict (5).

Now suppose the assumption is true. Define functions $g^{(\cdot)}, h^{(\cdot)}$ by machines having oracle access such that for every $\mu \in A(\Phi \cap \Omega_s)$,

$$\big(g^\mu(x,0), g^\mu(x,1), \ldots, g^\mu(x,h^\mu(x))\big) = \begin{cases} [m]^{\hat{\mu}}(x) & \text{if } \sigma(x) \notin D_{\beta_{m-1}}, \\ \vec{w} \in \Gamma \text{ with } \tau_\mu(x) \simeq G(\vec{w}) & \text{otherwise,} \end{cases} \tag{6}$$

where $\hat{\mu}(z)$ is set to $\mu(z)$ if $z \notin D_{\beta_{m-1}}$ and to $\beta_{m-1}(z)$ otherwise. Note that $\hat{\mu}$ is a total extension of β_{m-1}, and if $\mu \in A(\Phi \cap \Omega_s)$ then also $\hat{\mu} \in A(\Phi \cap \Omega_s)$. We have to show that $g^\mu, h^\mu \in \mathrm{FP}^\mu$ for every $\mu \in A(\Phi \cap \Omega_s)$. Since $\sigma \in \mathrm{FP}$ and $D_{\beta_{m-1}}$ is finite the cases in the definition in equation (6) are distinguishable in polynomial time. Hence, the first case can be treated in polynomial time. For the other case note that the set $\{\, \mu|_{D_{\beta_{m-1}}} \mid \mu \in A(\Phi \cap \Omega_s) \,\}$ is finite because $D_{\beta_{m-1}}$ is finite and since the function values of μ are restricted by s ($\mu \in A(\Phi \cap \Omega_s) \subseteq A(\Omega_s)$). Therefore there exists a table assigning to every possible oracle $\mu|_{D_{\beta_{m-1}}}$ a $\vec{w}$ as required in the definition. The existence of such a $\vec{w}$ is given by fact (3). Thus, for every $\mu \in A(\Phi \cap \Omega_s)$, $g^\mu, f^\mu \in \mathrm{FP}^\mu$.

For every function $\mu \in A(\Phi \cap \Omega_s)$ and for every $x \in \Sigma^*$, we now consider two cases:

Case 1. $\sigma(x) \notin D_{\beta_{m-1}}$. But then $\big(g^\mu(x,0), g^\mu(x,1), \ldots, g^\mu(x,h^\mu(x))\big) = [m]^{\hat{\mu}}(x) \in \Gamma$ because of the first conjunct in assumption (5), and

$\tau_\mu(x)$	$\simeq \tau_{\hat{\mu}}(x)$	by definition of $\tau_\mu, \tau_{\hat{\mu}}$, and observing that, if $\sigma(x) \notin D_{\beta_{m-1}}$ then $\sigma(x) + i \notin D_{\beta_{m-1}}$ for all $1 \leq i \leq x$,
	$\simeq G([m]^{\hat{\mu}}(x))$	by assumption (5) second conjunct,
	$\simeq G\big(g^\mu(x,0), g^\mu(x,1), \ldots, g^\mu(x,h^\mu(x))\big)$	because we are in case 1.

Case 2. $\sigma(x) \in D_{\beta_{m-1}}$. Then, by definition, we have $\big(g^\mu(x,0), g^\mu(x,1), \ldots, g^\mu(x,h^\mu(x))\big) \in \Gamma$ and $\tau_\mu(x) = G\big(g^\mu(x,0), g^\mu(x,1), \ldots, g^\mu(x,h^\mu(x))\big)$.

Thus we see that there exists an $m_0 \in \mathbb{N}$ (the index of the pair of machines realizing $g^{(\cdot)}$ and $h^{(\cdot)}$ in our enumeration) such that for every $\mu \in A(\Phi \cap \Omega_s)$ and for every $x \in \Sigma^*$ we have $[m_0]^\mu(x) \in \Gamma$ and $\tau_\mu(x) \simeq G([m_0]^\mu(x))$. Clearly, this is a contradiction to assumption (4). Thus either assumption (4) is false (and then the claim is already proven) or assumption (5) is false.
End of Stage m.

This completes the proof of the theorem. ❑

With this theorem we state a result similar to [9, 36] for function classes. The following corollary, which is very important for the applications, is immediate.

Corollary 5.2. *Let (F, Φ) be a paddable generator, and let (G, Γ) be a generator. If there exists a total function s such that $(F, \Phi) \not\leq_{s\text{-}m}^{\mathrm{FPLT}} (G, \Gamma)$, then there exists an α such that $(F, \Phi)\text{-FP}^{\alpha} \not\subseteq (G, \Gamma)\text{-FP}^{\alpha}$.*

Thus to separate two function classes all we have to do is to find a function s for which one of the generators does not s-partially reduce to the other.

Finally let us mention that we talked about separability via oracle functions above. However, in our model there is no difference between oracle separations by functions and oracle separations by sets. To be more precise:

Proposition 5.3. *Let (F, Φ) and (G, Γ) be generators. Then the following statements are equivalent:*

1. $(F, \Phi)\text{-FP}^{\alpha} \subseteq (G, \Gamma)\text{-FP}^{\alpha}$ *for every polynomially length-bounded function α.*
2. $(F, \Phi)\text{-FP}^{\alpha} \subseteq (G, \Gamma)\text{-FP}^{\alpha}$ *for every function α.*
3. $(F, \Phi)\text{-FP}^{A} \subseteq (G, \Gamma)\text{-FP}^{A}$ *for every $A \subseteq \Sigma^*$.*

Proof. That *1* implies *2* is a consequence of our computation model—our machines can read oracle values only up to a polynomial length. *2* implies *1* trivially.

Equivalence of *1* and *3* follows by identifying a function f with its projection $\mathrm{proj}(f)$, where $\mathrm{proj}(f) = \{(x, y) \mid f(x) \geq y\}$. Since $\mathrm{FP}^f = \mathrm{FP}^{\mathrm{proj}(f)}$ (by binary search), the equivalence is obvious. ❑

When we consider zero-one valued generator functions, i.e., classes of characteristic functions which can be identified with classes of sets, then our main theorem yields as a corollary the separability criterion given in [9, 36].

6 Applications

We give some applications of our criterion and show the existence of separating oracles. In all cases no structural consequences, e.g., via the *operator method*, are known to follow from the positive inclusions. First, we turn to the open question $\min\cdot \mathrm{P} \subseteq \#\cdot \mathrm{NP}$ from [14]. Our criterion easily yields a relativized world where the answer is negative. We remark that Theorem 6.1 achieves a similar result to the one proven by Glaßer and Wechsung [12] via a suitable diagonalization.

Theorem 6.1. *There is an oracle A such that* $\min\cdot \mathrm{P}^A \not\subseteq \#\cdot \mathrm{NP}^A$.

Proof. Suppose that such an oracle A does not exist, i.e., for $F_1(\vec{u}) =_{\text{def}} \min\{\, j \mid u_j > 0 \,\}$, $F_2(\vec{u}) =_{\text{def}} \text{span}_+(\vec{u})$ (see Table 1) and for all total s we have $(F_1, \Omega) \leq_{s\text{-}m}^{\text{FPLT}} (F_2, \Omega)$ by Theorem 5.1 and Proposition 5.3. Let $s(n) =_{\text{def}} 1$ for all $n \in \mathbb{N}$ and $g, h \in \text{FPLT}$ be functions witnessing the reduction. Let p be a polynomial such that $g(\vec{u}, j)$ can be computed in run-time $p(\log(|\vec{u}| + |j|))$ and $h(\vec{u})$ can be computed in run-time $p(\log|\vec{u}|)$. Note that we can replace $|\vec{u}|$ by $\dim \vec{u}$.

We start with a vector $\vec{u}_0 =_{\text{def}} (0, 0, \ldots, 0, 1) \in \Omega_s$ having k components, i.e., $F_1(\vec{u}_0) = k = \text{span}_+(g(\vec{u}, 0), \ldots, g(\vec{u}, h(\vec{u})))$. Let k be large enough to satisfy $k \geq p(\log k)$. Let $I_{\vec{u}_0}$ be the set of the k smallest indices $j \leq h(\vec{u}_0)$ such that $g(\vec{u}_0, j) > 0$ and $g(\vec{u}_0, i) \neq g(\vec{u}_0, j)$ for all $i < j$. For convenience, we assume that during the computation of $h(\vec{u}_0)$ only the k–th and the $p(\log k)$–leftmost components of the input are asked. (The general case can be proven similarly, but is technically more involved.) For $1 + p(\log k) \leq i \leq k - 1$, consider vectors $\vec{u}_i$ that are the same as $\vec{u}_0$ with the only difference that an additional 1 occurs in the i-th component of $\vec{u}_i$. Then it holds that $|\vec{u}_0| = |\vec{u}_i|$, $h(\vec{u}_0) = h(\vec{u}_i)$, and, since $\text{span}_+(g(\vec{u}, 0), \ldots, gq(\vec{u}, h(\vec{u}))) = i$, there are at least $k - i$ indices $j \in I_{\vec{u}_0}$ with $g(\vec{u}_i, j) \neq g(\vec{u}_0, j)$ to realize the decreased value of span_+. Hence, the computations of $g(\vec{u}_i, j)$ for all these j have to access the i-th components of $\vec{u}_i$. Overall, in order to distinguish $\vec{u}_0$ from any $\vec{u}_i$ for $1 + p(\log k) \leq i \leq k - 1$, the necessary amount of accesses to a given k-dimensional input vector from Ω_s is

$$\sum_{i=1+p(\log k)}^{k} k - i \;=\; \frac{(k - p(\log k))^2 + k - p(\log k)}{2} \;\geq\; \frac{(k - p(\log k))^2}{2}.$$

On the other side, since $g \in \text{FPLT}$, for a given k-dimensional input vector, the largest possible number of accesses to different vector components is $k \cdot p(\log(k + \log k))$ which is asymptotically strictly less than $\frac{1}{2}(k - p(\log k))^2$. Hence, the functions g and h do not s-partially reduce (F_1, Ω) to (F_2, Ω). Since the arguments do not depend on the choice of g, h, we have a contradiction to our assumption. ❑

For our further separation examples, we return to the classes $\text{c\#}\cdot\text{coNP}$ and $\text{c\#}\cdot\text{P}^{\text{NP}}$, defined and characterized in Subsect. 3.2 (see Table 2). These classes are related to other well-known function classes in [22]. However, some questions remained open. First, it is not known if $\max\cdot\text{NP} \subseteq \text{c\#}\cdot\text{coNP}$ (observe that $\max\cdot\text{NP} \subseteq \#\cdot\text{coNP}$ [20, 21]). We now construct a relativized world where the answer is no.

Theorem 6.2. *There is an oracle A such that* $\max\cdot\text{NP}^A \not\subseteq \text{c\#}\cdot\text{coNP}^A$.

Proof. Again, suppose that such an oracle A does not exist, so for the generators (F_1, Ω) and (F_2, Φ_2) of $\max\cdot\text{NP}$ and $\text{c\#}\cdot\text{coNP}$, respectively, from Table 1 and for all s we have $(F_1, \Omega) \leq_{s\text{-}m}^{\text{FPLT}} (F_2, \Phi_2)$. Let $s(n) =_{\text{def}} 2^n$ and $g, h \in \text{FPLT}$ witness the reduction for all $\vec{u} \in \Omega_s$. We start with a vector $\vec{u}_0 =_{\text{def}} (10, 10, \ldots, 10, 0, \ldots, 0, 0)$ having l components '10', which is the value of 2 in binary, and a total of k components (we will fix l soon). Then, the set of values in $\big(g(\vec{u}_0, 0), g(\vec{u}_0, 1), \ldots, g(\vec{u}_0, h(\vec{u}))\big)$ is a set $[1, max]$ with $max = g(\vec{u}_0, 0)$ and having a gap of two consecutive values. Without loss of generality we can assume that max is odd. So there exists an index j such that $g(\vec{u}_0, j) \in \big\{\frac{max-1}{2}, \frac{max-1}{2} + 1, \frac{max-1}{2} + 2\big\}$.

Next, we will turn $\vec{u}_0$ into a vector $\vec{u}_c$ for some c by placing c in one of the components where we had a '0' before while switching a certain number of '10' entries to '0' to preserve the length of the vector. This increases the gap from size two to c. Now, the contradiction for the computation $g(\vec{u}_0, j)$ follows if we can ensure by appropriate choices for l, c, and the components we switch that

1. $|\vec{u}_0| = |\vec{u}_c|$, $g(\vec{u}_0, 0) = max = g(\vec{u}_c, 0)$ and $h(\vec{u}_0) = h(\vec{u}_c)$,
2. there are super-polylogarithmic many positions to place c in $\vec{u}_c$, and
3. $g(\vec{u}_0, j)$ is always in the gap of size c—no matter where the gap is in $[1, max]$—so it must be that $g(\vec{u}_0, j) \neq g(\vec{u}_c, j)$.

This can be achieved by choosing $c =_{\mathrm{def}} \frac{max-1}{2} + 2$, $l = |c| + t$ where t is the number of components accessed during the computations of $g(\vec{u}_0, 0)$, $h(\vec{u}_0)$, and $g(\vec{u}_0, j)$, and switching only components not accessed during these computations. Note that $\vec{u}_c \in \Omega_s$. ❑

A second open question in [22] is if $\mathrm{c}\#\cdot \mathrm{coNP} = \mathrm{c}\#\cdot \mathrm{P}^{\mathrm{NP}}$. Again, note that in the non-cluster world the answer is known: $\#\cdot \mathrm{coNP} = \#\cdot \mathrm{P}^{\mathrm{NP}}$ as we already mentioned in Subsect. 3.2. From the preceding theorem we now obtain again a relativized world where the question is answered to the negative.

Corollary 6.3. *There is an oracle A such that* $\mathrm{c}\#\cdot \mathrm{coNP}^A = \mathrm{c}\#\cdot \mathrm{P}^{\mathrm{NP}^A}$.

Proof. In [22] it was shown that (relativizably) $\max \cdot \mathrm{NP} \subseteq \mathrm{c}\#\cdot \mathrm{P}^{\mathrm{NP}}$. Hence, the corollary follows immediately from Theorem 6.2. ❑

7 Conclusion

We presented a framework for the uniform definability of function classes. With a number of examples we made clear that virtually all (polynomial-time) function classes being of current topical interest in complexity theory can be defined using our machinery. This allows us to contribute to the study of the relations between such function classes, either from an oracle separation point of view or from a more structural point of view.

Besides finding more characterizations it will of course be interesting to see if our criterion will turn out to be as useful for the examination of function classes as the leaf language criterion from [9, 36] was in the case of classes of sets.

Acknowledgment. Thanks are due to Ulrich Hertrampf (Stuttgart), Klaus W. Wagner (Würzburg), and the anonymous referees for helpful hints.

References

[1] E. Allender. Oracles versus proof techniques that do not relativize. In *Proceedings 1st International Symposium on Algorithms*, volume 450 of *Lecture Notes in Computer Science*, pages 39–52. Springer Verlag, 1990.

[2] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 2nd edition, 1995.

[3] R. V. Book, T. J. Long, and A. L. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13:461–487, 1984.

[4] B. Borchert. *Predicate classes, promise classes, and the acceptance power of regular languages.* PhD thesis, Naturwissenschaftlich-Mathematische Fakultät, Universität Heidelberg, 1994.

[5] B. Borchert. On the acceptance power of regular languages. *Theoretical Computer Science*, 148:207–225, 1995.

[6] B. Borchert and A. Lozano. Succinct circuit representations and leaf language classes are basically the same concept. *Information Processing Letters*, 58:211–215, 1996.

[7] B. Borchert and R. Silvestri. The general notion of a dot operator. In *Proceedings 12th Conference on Computational Complexity*, pages 26–44. IEEE Computer Society Press, 1997.

[8] D. P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* International Series in Computer Science. Prentice Hall, London, 1994.

[9] D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104:263–283, 1992.

[10] S. A. Fenner, L. Fortnow, and S. A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

[11] L. Fortnow. The role of relativization in complexity theory. *Bulletin of the EATCS*, 52:229–244, 1994.

[12] C. Glaßer and G. Wechsung. Relativizing function classes. Technical Report 219, Institut für Informatik, Universität Würzburg, 1999.

[13] L. A. Hemaspaandra and H. Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *Complexity Theory Column 8, ACM-SIGACT Newsletter*, 26(1):2–13, 1995.

[14] H. Hempel and G. Wechsung. The operators min and max on the polynomial time hierarchy. In *Proceedings 14th Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *Lecture Notes in Computer Science*, pages 93–104. Springer Verlag, 1997.

[15] U. Hertrampf. Classes of bounded counting type and their inclusion relations. In *Proceedings 12th Symposium on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 60–70. Springer Verlag, 1995.

[16] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.

[17] U. Hertrampf, H. Vollmer, and K. W. Wagner. On the power of number-theoretic operations with respect to counting. In *Proceedings 10th Structure in Complexity Theory*, pages 299–314, 1995.

[18] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Information & Computation*, 129:21–33, 1996.

[19] B. Jenner and J. Torán. Computing functions with parallel queries to NP. *Theoretical Computer Science*, 141:175–193, 1995.

[20] J. Köbler. *Strukturelle Komplexität von Anzahlproblemen.* PhD thesis, Fakultät für Informatik, Universität Stuttgart, 1989.

[21] J. Köbler, U. Schöning, and J. Torán. On counting and approximation. *Acta Informatica*, 26:363–379, 1989.

[22] S. Kosub. A note on unambiguous function classes. *Information Processing Letters*, 72(5-6):197–203, 1999.

[23] S. Kosub, H. Schmitz, and H. Vollmer. Uniformly defining complexity classes of functions. In *Proceedings 15th Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*, pages 607–617. Springer Verlag, 1998.

[24] M. W. Krentel. The complexity of optimization functions. *Journal of Computer and System Sciences*, 36:490–509, 1988.

[25] M. W. Krentel. Generalizations of OptP to the polynomial hierarchy. *Theoretical Computer Science*, 97:183–198, 1992.

[26] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

[27] C. H. Papadimitriou. Personal communication, 1994.

[28] C. H. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings 6th GI Conference on Theoretical Computer Science*, volume 145 of *Lecture Notes in Computer Science*, pages 269–275. Springer Verlag, 1983.

[29] K. Regan and H. Vollmer. Gap-languages and log-time complexity classes. *Theoretical Computer Science*, 188:101–116, 1997.

[30] H. Schmitz. Nichtdeterministische Polynomialzeit-Berechnung von Funktionen. Master's thesis, Institut für Informatik, Universität Würzburg, 1996.

[31] A. L. Selman. A taxonomy on complexity classes of functions. *Journal of Computer and System Sciences*, 48:357–381, 1994.

[32] S. Toda. *Computational Complexity of Counting Complexity Classes.* PhD thesis, Department of Computer Science, Tokyo Institute of Technology, Tokyo, 1991.

[33] S. Toda. PP is as hard as the polynomial time hierarchy. *SIAM Journal on Computing*, 20:865–877, 1991.

[34] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[35] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8(3):411–421, 1979.

[36] N. K. Vereshchagin. Relativizable and non-relativizable theorems in the polynomial theory of algorithms. *Izvestija Rossijskoj Akademii Nauk*, 57:51–90, 1993. In Russian.

[37] H. Vollmer. On different reducibility notions for function classes. In *Proceedings 11th Symposium on Theoretical Aspects of Computer Science*, volume 775 of *Lecture Notes in Computer Science*, pages 449–460. Springer Verlag, 1994.

[38] H. Vollmer. Uniform characterizations of complexity classes. *Complexity Theory Column 23, ACM-SIGACT Newsletter*, 30(1):17–27, 1999.

[39] H. Vollmer and K. W. Wagner. The complexity of finding middle elements. *International Journal of Foundations of Computer Science*, 4:293–307, 1993.

[40] H. Vollmer and K. W. Wagner. Complexity classes of optimization functions. *Information & Computation*, 120:198–219, 1995.