Relativization and Interactive Proof Systems in Parameterized Complexity Theory*†

Ralph Christian Bottesch

QuSoft, CWI, Amsterdam, The Netherlands
botteschecwi.nl

Abstract

We introduce some classical complexity-theoretic techniques to Parameterized Complexity. First, we study relativization for the machine models that were used by Chen, Flum, and Grohe (2005) to characterize a number of parameterized complexity classes. Here we obtain a new and non-trivial characterization of the **A**-Hierarchy in terms of oracle machines, and parameterize a famous result of Baker, Gill, and Solovay (1975), by proving that, relative to specific oracles, **FPT** and **A**[**1**] can either coincide or differ (a similar statement holds for **FPT** and **W**[**P**]). Second, we initiate the study of interactive proof systems in the parameterized setting, and show that every problem in the class **AW**[**SAT**] has a proof system with "short" interactions, in the sense that the number of rounds is upper-bounded in terms of the parameter value alone.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes

Keywords and phrases Parameterized complexity, Relativization, Interactive proofs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2017.9

1 Introduction

In Parameterized Complexity Theory, the complexity of computational problems is measured not only in terms of the size of the input, $|x|$, but also in terms of a parameter k which measures some additional structure of the input. The main advantage of this approach is that the class of problems which are considered computationally tractable can be expanded considerably by requiring that the running time of algorithms be polynomial only in $|x|$, while allowing some other dependence of the running time on the parameter value. Problems that can be solved by such algorithms are said to be *fixed-parameter tractable*. To this relaxed notion of computational tractability there corresponds a matching notion of intractability.

The complexity classes capturing parameterized intractability were originally defined as closures, under suitably defined parameterized reductions, of specific problems that were conjectured to not have fpt-algorithms (see [8], or the more recent [9]). This approach ensured that most of these "hard" classes contained an interesting or somewhat natural complete problem, and, in the case of **W**[1], produced a "web of reductions" similar to the one for **NP**-complete problems in classical complexity.

However, defining complexity classes only via reductions to specific problems means that the resulting classes may not have characterizations in terms of computing machines, or, indeed, any natural characterizations except the definition. This in turn can mean that many proof techniques from classical complexity are not usable in the parameterized setting,

* This work is supported by the ERC Consolidator Grant QPROGRESS 615307.
† A full version of the paper is available at https://arxiv.org/abs/1706.09391.

12th International Symposium on Parameterized and Exact Computation (IPEC 2017).
Editors: Daniel Lokshtanov and Naomi Nishimura; Article No. 9; pp. 9:1–9:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

because they rely on different characterizations that do not apply to any one parameterized complexity class. To give an example, in the proof of **IP** = **PSPACE** ([15], see also [16]), both the definition of **PSPACE** in terms of space-bounded computation, and the characterization of this class in terms of alternating polynomial-time computation are used. In the parameterized world, this equivalence between space and alternating time seems to break down [6], and parameterized interactive proof systems do not appear to have been studied at all, so no similar theorem is known in this setting.

Surprisingly (given the way they were originally defined), many of the classes capturing parameterized intractability turned out to have characterizations in terms of computing machines: In three papers, Chen [5, 6, 7], Flum [5, 6, 7], and Grohe [6, 7] showed that certain kinds of nondeterministic random access machines (RAMs) exactly define some important parameterized classes:

- **W**[**P**] and **AW**[**P**] are characterized by RAMs that can nondeterministically [1] guess integers, but the number of guesses they can make throughout the computation is bounded by a computable function of the parameter value of the input instance. We refer to this as *parameter-bounded nondeterminism* (a term used similarly in [6]).
- The classes of the **A**-Hierarchy, as well as **AW**[∗], are obtained by further restricting the (alternating) nondeterminism of the machines to *tail-nondeterminism*, meaning that the machines can only make nondeterministic guesses among the last $h(k)$ steps of a computation, where h is a computable function and k is the parameter.
- Finally, the classes of the **W**-Hierarchy are characterized by tail-nondeterministic machines which are not allowed to access the guessed integers directly (they can make nondeterministic decisions based on them, but not use them in arithmetic operations).

The main reason why the characterizations in [5, 6, 7] were given in terms of RAMs, rather than Turing machines (TMs), is that a TM may need to traverse the entire used portion of its tape in order to read a particular bit, so a tail-nondeterministic TM would not be able to make use of its entire memory during the nondeterministic phase of the computation. The classes **W**[**P**] and **AW**[**P**] also have characterizations in terms of TMs with restricted nondeterminism [6], but we consistently use random access machines throughout this work.

The machine characterizations of some of the above-mentioned classes can be rewritten in such a way that they strongly resemble definitions of some familiar classes from classical complexity. For example, **A**[1] can be defined as the class of parameterized problems that are decided by tail-nondeterministic RAMs in *fpt-time*, which at least formally looks like the definition of **NP**. Similarly, **W**[**P**] can also be defined in a way that is similar to **NP** (using parameter-bounded nondeterminism), the levels of the **A**-Hierarchy have characterizations that match the definitions of the Σ-levels of the Polynomial Hierarchy, and **AW**[**P**] and **AW**[∗] both correspond to **AP** (the class of problems that are decidable in alternating polynomial-time). Given the similar definitions, it seems reasonable to expect that parameterized complexity classes also inherit some properties from their classical counterparts. However, replacing the machine model in a definition is a significant change, so it is by no means obvious which theorems will still hold for a parameterized version of a complexity class.

Our goal in this paper is to show that having machine-based characterizations of parameterized complexity classes opens up a largely unexplored, but possibly very fruitful, path toward understanding parameterized intractability. To that end we extend the work

[1] Throughout this paper, nondeterminism will mean alternating nondeterminism with a number of alternations that will be clear from the context. This should not cause any confusion, since simple nondeterminism is just 1-alternating nondeterminism.

of Chen, Flum, and Grohe [5, 6, 7] in two directions: relativization and interactive proofs. The key insight is that parameterized versions of these two concepts can be defined in such a way that some important classical theorems can be recovered in this setting. The proofs of our theorems follow along the same lines as their classical counterparts, with only some technical obstacles to be overcome, but it is a remarkable fact that parameterized versions of these proofs can be made to work at all: For example, it is not a priori clear whether parameterized oracle computation can be even in principle defined in a way that makes the **A**-Hierarchy have an oracle characterization that is similar to that of **PH**. We show, among other things, that this is indeed the case, and furthermore, that the restrictions that must be placed on the access to the oracle in order to obtain this result are quite natural (at least, in the context of the machine characterization of **A**[1] from [7]).

1.1 Our results

Parameterized relativization. Theorems involving oracles have been given before in Parameterized Complexity, but it is almost always Turing machines that are endowed with access to an oracle (see, for example, [13]). In order to relativize the hard parameterized complexity classes for which machine characterizations are known, we define oracle RAMs with the different forms of restricted nondeterminism mentioned above. It turns out that in order for oracle access and nondeterminism to interact in a useful way, both of these features must, roughly speaking, have the same restrictions (tail-nondeterministic machines should have tail-restricted oracle access, etc.)[2]. We show that these restrictions lead to a natural type of oracle access for each type of machine, by proving parameterized versions of two fundamental results from classical complexity, both for the tail-nondeterministic and the parameter-bounded version of nondeterministic RAMs.

First, we give a new characterization of the classes of the **A**-Hierarchy, in terms of oracle machines (resembling the oracle characterization of the levels of the Polynomial Hierarchy (see [3], Section 5.5)), by proving that

$$\forall t \geq 1 : \mathbf{A}[1]^{O_t} = \mathbf{A}[t+1],$$

but only for a *specific* oracle O_t that is complete for $\mathbf{A}[t]$ (Theorem 13). We also explain why tail-nondeterminism appears to be too weak to allow for this theorem to be proved for an arbitrary $\mathbf{A}[t]$-complete problem. The situation is much better when the nondeterminism is only parameter-bounded, and we have (Theorem 16) that

$$\forall t \geq 1 : \mathbf{W}[\mathbf{P}]^{\mathbf{\Sigma}_t^{[P]}} = \mathbf{\Sigma}_{t+1}^{[P]},$$

where $\mathbf{\Sigma}_t^{[P]}$ $(t \geq 1)$ are the Σ-levels of the analogue of the Polynomial Hierarchy for the machine model with parameter-bounded nondeterminism (so $\mathbf{\Sigma}_1^{[P]} = \mathbf{W}[\mathbf{P}]$). We emphasize that both of these theorems seem to hold only if the oracle $\mathbf{A}[1]$- and $\mathbf{W}[\mathbf{P}]$-machines have exactly the right restrictions placed on their oracle access, and even then, tail-nondeterminism causes a number of non-trivial technical issues (see the proof of Theorem 13).

---

[2] Placing restrictions on the access to an oracle is a fairly common practice even in classical complexity. For example, the oracle tape of a **LOGSPACE**-machine is write-only, in order to allow the machine to make polynomial-sized queries while preventing it from using the tape for computations that avoid the space restriction. Another example can be found in [1], where, in order to prove that the statement $\mathbf{NEXP} \subset \mathbf{MIP}$ *algebrizes*, the authors restrict machines that run in exponential time so that they can only make oracle queries of polynomial size.

Second, we recover a parameterized version of a well-known oracle separation result of Baker, Gill, and Solovay [4], by showing (Theorem 14) that there exist parameterized oracles A and B such that

$$\mathbf{FPT}^A = \mathbf{A}[1]^A \quad \text{and} \quad \mathbf{FPT}^B \neq \mathbf{A}[1]^B.$$

It is worth noting that here the **FPT**-machine may be given completely unrestricted access to the oracle B, whereas the **A**[1]-machine only has tail-restricted access (which is the most restricted form of oracle access we consider), so in some sense this separation is stronger than expected. A similar theorem holds when replacing **A**[1] with **W**[**P**] (Theorem 18).

These results are, of course, only the first steps toward understanding relativization for parameterized complexity classes beyond **FPT**. To illustrate the importance of investigating relativization in this setting, let us briefly consider the long-standing open problem of proving a parameterized version of Toda's Theorem [17], which states that $\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{PP}}$. It is not clear which parameterized classes would be involved in such a theorem, but, presumably, **P** would be replaced by **FPT**, which can easily be described in terms of Turing machines, so it should be possible to at least state the theorem without further considerations about the type of oracle access being used. Furthermore, it could be argued that since only the larger of the two classes in the theorem statement is obtained via relativization, placing no restrictions on the access to the oracle can only make the inclusion easier to prove. However, both Toda's original proof [17] and Fortnow's simplified version of it [12] make heavy use of relativized versions of classes such as **BPP** and **PH**, so following either one of these proofs would involve relativized versions of parameterized counterparts of such classes. Our Theorems 13 and 16 only deal with oracle access and alternating nondeterminism, but this already requires a careful balancing of the restrictions placed on both features. Toda's Theorem, on the other hand, involves an interplay between relativization, alternating nondeterminism, randomization, and counting complexity, so it seems unlikely that a parameterized version of it can be proved without a better understanding of parameterized relativization and its relation to other complexity-theoretic concepts.

Interactive proof systems for parameterized complexity classes. The levels of the **A**-Hierarchy were originally defined as fpt-closures of *model checking* problems, where a relational structure $\mathcal{A}$ and a first-order formula ϕ without free variables are given, and the task is to decide whether $\mathcal{A}$ satisfies ϕ. In [7], model checking problems are used in a very interesting way in the proof of the machine characterization of the classes $\mathbf{A}[t]$: Specifically, a pair $(\mathcal{A}, \phi)$ is used to encode the computation of a tail-nondeterministic RAM, in a way that is strongly reminiscent of how the computation of a nondeterministic TM is encoded as a quantified Boolean formula in the proof of the Cook-Levin Theorem (see [3], Chap. 2). This suggests that by generalizing classical techniques that involve quantified Boolean formulas, it may be possible to apply them to parameterized complexity classes for which a model checking problem is complete. In Section 4 we continue this line of thought by generalizing *arithmetization* of quantified Boolean formulas (see [3], Section 8.3) to pairs of relational structures and first-order formulas.

We also initiate the study of interactive proof systems in this setting. Using generalized arithmetization, we show that all problems in **AW**[**SAT**] have proof systems with a number of rounds depending only on the parameter value of the input instance (Theorem 19). The goal (which, unfortunately, is not achieved here) is to precisely characterize either $\mathbf{AW}[*]$ or **AW**[**P**] in terms of IPs, as this would recover a parameterized version of the fact that $\mathbf{IP} = \mathbf{AP}$, even without a notion of space that corresponds to alternation in the parameterized setting. At the end of Section 4 we give a possible candidate for a characterization of $\mathbf{AW}[*]$.

Note that theorem proofs and other details can be found in the full version of paper (arXiv:1706.09391).

2 Preliminaries

We refer to [3] and to [11], respectively, for the necessary background in classical and Parameterized Complexity. By $\mathbb{N}$ we mean the set of non-negative integers, and by $\mathbb{N}^*$ the set of finite sequences of non-negative integers.

2.1 Random access machines and parameterized complexity classes

We give only a general overview of RAMs, and refer to Section 2.6 of [14] for the details. A random access machine is specified by its *program* (a finite sequence of instructions), which operates on an infinite sequence of *standard registers*, $r_0, r_1, \ldots$, that contain integers. Instructions access registers either directly, by referencing their numbers, or indirectly, by taking the number of a register to be the current content of another register (in other words, the machine can access r_{r_i}, $i \in \mathbb{N}$). We follow [6] in assuming that the registers store only non-negative integers. Except instructions which copy the contents of one register to another, a RAM also has conditional and unconditional jump instructions, as well as instructions which perform the operations addition, subtraction, and integer division by 2 (these suffice to efficiently perform all arithmetic operations on signed integers). The input of a RAM is a finite sequence of non-negative integers, each stored in a separate register, and we define the problems solved by such machines accordingly.

▶ **Definition 1.** A *parameterized problem* Q is a subset of $\mathbb{N}^* \times \mathbb{N}$. When dealing with the problem of deciding whether $(x, k) \in \mathbb{N}^* \times \mathbb{N}$ is an element of Q, (x, k) is referred to as an *instance*; the second element of such a pair is called the *parameter*.

▶ **Remark 2.** When an instance of a parameterized problem is given as input to a RAM, we assume that the parameter is given in unary encoding, meaning that if the parameter value is $k \in \mathbb{N}$, then k registers, each containing the value 1, are used to encode the parameter value. The size of x, the main part of the input, is taken as the sum of the sizes of the binary encodings of the integers that make up x. A RAM can therefore efficiently convert between a reasonable encoding using integers, and any reasonable encoding using a finite alphabet.

▶ **Definition 3.** A random access machine $\mathbb{M}$ is *parameter-restricted* if there is a computable function f and a polynomial function p, such that on any input (x, k):
- $\mathbb{M}$ terminates after executing at most $f(k)p(|x|)$ instructions;
- throughout any computation, the registers contain only numbers that are $\leq f(k)p(|x|)$.

The above definition replaces the "polynomial-time" restriction on the running time in the classical setting, and is similar to the definition of "κ-restricted" in Chap. 6 of [11]. Note that the second condition is a bound on the numbers stored in the registers, not on the number of bits that would be needed for the binary encoding of these numbers.

The next definition is easily seen to be equivalent to the usual definition of **FPT** [11].

▶ **Definition 4.** We define **FPT** as the class of parameterized problems that are decidable by parameter-restricted (deterministic) RAMs.

An *alternating random access machine (ARAM)* is a RAM with additional *existential* and *universal guess instructions*, EXISTS and FORALL, both of which place a nondeterministically chosen integer from the interval $[0, r_0]$ into r_0 (the difference between the two instructions is in how the acceptance of the input is defined). We may assume that the upper end of the range of each nondeterministic guess is the largest number that the machine can store in its registers, given the input, because the machine can first guess a number in the maximum range, and then trim the result by computing the remainder of a division by the size of the intended range. For ARAMs, the notions of computation (on an input), configuration, computation path, t-alternation, and acceptance/rejection of an input are defined in the standard way (see [11], section 8.1, pp. 168-170). Following [7], we mean by "t-alternating" that the first guess instruction is existential.

We give the definitions of some complexity classes in terms of nondeterministic RAMs. These are not the original definitions, but characterizations proved in [6] and [7].

▶ **Definition 5.** A parameterized problem Q is in $\mathbf{AW}[\mathbf{P}]$ [in $\mathbf{W}[\mathbf{P}]$] if it is decided by an ARAM [a 1-alternating ARAM] $\mathbb{A}$ which, for some computable function h, on any input (x, k), executes at most $h(k)$ nondeterministic instructions on any computation path.

▶ **Definition 6.** An ARAM $\mathbb{A}$ is *tail-nondeterministic* if there is a computable function g such that, on any input (x, k), $\mathbb{A}$ executes nondeterministic instructions only among the last $g(k)$ steps of any computation path. For every $t \geq 1$, $\mathbf{A}[t]$ denotes the class of parameterized problems that are decidable by parameter-restricted tail-nondeterministic t-alternating ARAMs. $\mathbf{AW}[*]$ denotes the class of parameterized problems that are decidable by parameter-restricted tail-nondeterministic ARAMs.

An *oracle (A)RAM* or *(A)RAM with access to an oracle* is a machine with an additional set of *oracle registers* that store non-negative integers, as well as instructions that copy the contents of r_0 to an arbitrary oracle register and vice-versa, and a QUERY instruction, which queries the oracle with the contents of the oracle registers, and causes the register r_0 to contain the values 1 or 0 (representing the oracle's answer). Note that we only work with oracles that decide parameterized problems, and that the parameter of a query instance must be encoded in unary (see Remark 2). Most previous results involving oracles in Parameterized Complexity place the following restriction on oracle machines. We will consider additional restrictions to oracle access in the next section.

▶ **Definition 7.** An oracle (A)RAM $\mathbb{A}$ has *balanced* access to an oracle if there is a computable function g such that, on input (x, k), any query (y, k') made to the oracle, on any computation path, satisfies $k' \leq g(k)$.

2.2 Relational structures and first-order formulas

A *relational vocabulary* τ is a set of pairs of symbols and positive integers, called *relational symbols* and *arities*, respectively. A *relational structure* $\mathcal{A}$ *with vocabulary* τ is a set containing: a set A, called the *universe of* $\mathcal{A}$, and for each pair $(s, r) \in \tau$, a relation $R^s \subseteq A^r$. We only use relational structures with finite universes and finite vocabularies, so we assume that $A = \{0, \dots, n\}$, $n \in \mathbb{N}$. A *first-order formula* ϕ *with vocabulary* τ is constructed in the same way as a quantified Boolean formula, except that the *atomic formulas* are not variables, but expressions of the form $x_1 = x_2$ or $R^s x_1 \dots x_r$, where $x_1, \dots, x_r$ are variables and $(s, r) \in \tau$.

Whenever a pair $(\mathcal{A}, \phi)$ is given, it is assumed implicitly that $\mathcal{A}$ and ϕ share the same relational vocabulary. We say that $\mathcal{A}$ *satisfies* ϕ if ϕ is true when all atomic formulas are evaluated based on the relations in $\mathcal{A}$ and all variables are taken as ranging over A.

We define some important classes of first-order formulas with relational vocabularies. For every $t \in \mathbb{N}$, let Σ_t be the set of all first-order formulas of the form

$$\exists x_{1,1} \dots \exists x_{1,k_1} \forall x_{2,1} \dots \forall x_{2,k_2} \dots\dots Q x_{t,1} \dots Q x_{t,k_t} : \psi(x_1, \dots, x_t),$$

where $\psi(x_1, \dots, x_t)$ is a quantifier-free formula (Q means $\exists$ if t is odd, $\forall$ if t is even). For all $t, r \in \mathbb{N}$, let $\Sigma_t[r]$ be the set of all Σ_t-formulas with vocabularies in which all arities are $\leq r$. Finally, let PNF be the set of all first-order formulas in *prenex normal form*, meaning that they are of the form $Q_1 x_1 \dots Q_t x_t : \psi(x_1, \dots, x_t)$, where $\psi(x_1, \dots, x_t)$ is a quantifier-free formula and $Q_1, \dots, Q_t \in \{\exists, \forall\}$.

For certain classes of formulas F, the following parameterized *model checking* problems are complete for various important complexity classes.

p-MC(F)	
Input:	$(\mathcal{A}, \phi)$, where $\mathcal{A}$ is a relational structure, $\phi \in F$.
Parameter:	$\lvert\phi\rvert$.
Problem:	Decide whether $\mathcal{A}$ satisfies ϕ.

p-*var*-MC(F)	
Input:	$(\mathcal{A}, \phi)$, where $\mathcal{A}$ is a relational structure, $\phi \in F$.
Parameter:	The number of variables in ϕ.
Problem:	Decide whether $\mathcal{A}$ satisfies ϕ.

▶ **Remark 8.** A relational structure can be represented by listing the elements of its universe, followed by the tuples in each relation. However, for a RAM to check whether some tuple $(a_1, \dots, a_r)$ is an element of some r-ary relation R^s may then take a number of steps that depends on $\|\mathcal{A}\| := |A| + |\tau| + \sum_{(s,r)\in\tau} |R^s| \cdot r$ (even if the elements of each relation are listed in lexicographic order, and binary search is used). To avoid this, we will assume, whenever $\mathcal{A}$ contains only relations of arity at most some fixed number l, that each r-ary relation ($r \leq l$) is stored as an $|A|^r$-size array of ones and zeroes, each number representing whether or not some element of A^r is a member of the relation. Furthermore, we will assume that the location of every such array is stored in a look-up table. This way, checking whether $(a_1, \dots, a_r) \in R^s$ only takes a *constant* number of operations for a RAM, at the cost of increasing the size of the representation of $\mathcal{A}$ in memory to $O(\text{poly}(\|\mathcal{A}\|))$ (since l is constant). This also means that adding and removing elements requires only constant time.

▶ **Definition 9.** Let Q and Q' be parameterized problems. An algorithm $\mathbb{R}$ is an *fpt-reduction from Q to Q'* if there exist computable functions f and g, and a polynomial function p, such that for any instance (x, k) of Q we have a) $(y, k') := \mathbb{R}(x, k) \in Q'$ if and only if $(x, k) \in Q$; b) $\mathbb{R}$ runs in time $f(k)p(|x|)$; and c) $k' \leq h(k)$.

For any parameterized problem Q, we denote by $[Q]^{\text{fpt}}$ the set of parameterized problems that are $\leq^{\text{fpt}} Q$, meaning fpt-reducible to Q.

▶ **Fact 10** ([6, 10],[2]). For every $t \in \mathbb{N}$, $\mathbf{A}[t] = [p\text{-MC}(\Sigma_t)]^{\text{fpt}} = [p\text{-MC}(\Sigma_t[3])]^{\text{fpt}}$.
$\mathbf{AW}[\mathbf{SAT}] = [p\text{-}var\text{-MC(PNF)}]^{\text{fpt}}$.

▶ **Remark 11.** In the proof of their machine-based characterization of $\mathbf{A}[t]$, Chen, Flum, and Grohe [7] show how the parameter-restricted computation of a t-alternating tail-nondeterministic RAM can be encoded as a pair $(\mathcal{A}, \phi)$. We refer the interested reader to [7] for the details, and recall only some facts about this reduction that we use here. Let $f(k)p(|x|)$ be an upper bound on the running time, the largest number of a register used, and the largest integer stored during the computation of the machine $\mathbb{A}$ on input (x, k). The relational structure $\mathcal{A}$ has universe $\{0, \dots, f(k)p(|x|)\}$ and contains relations representing the instructions of $\mathbb{A}$'s program and the contents of the accessed registers at the end of the deterministic part of the computation (a relation *Reg* is defined so that $(y, z) \in Reg$ if and only if $r_y = z$ right before the first nondeterministic instruction is executed). All relations in $\mathcal{A}$ have arity ≤ 3. The first-order formula ϕ has the same vocabulary as $\mathcal{A}$ and encodes the nondeterministic computation of $\mathbb{A}$ (the last $h(k)$ steps). The formula is constructed in such a way that changes to the contents of the registers are kept track of, and access to the contents of the registers at the start of the nondeterministic computation are encoded using the relation Reg. A close look at the construction in [7] reveals that part of it is oblivious to the input x, in the sense that computing the formula ϕ only requires knowledge of k, $\mathbb{A}$.

3 Parameterized relativization

The guiding principle in our approach to defining nondeterministic oracle RAMs will be that all of the special resources of a machine (nondeterminism, oracle queries, random guesses – everything beyond the basic deterministic operations) should be restricted in the same way, in order for these resources to interact well with each other.

▶ **Definition 12.** An oracle (A)RAM $\mathbb{A}$ has *parameter-bounded* access to an oracle if it has balanced access to the oracle, and there is a computable function h such that, on input (x, k), $\mathbb{A}$ makes at most $h(k)$ queries to the oracle on any computation path. $\mathbb{A}$ is said to have *tail-restricted* access to an oracle if it has balanced access to the oracle, and there is a computable function h such that, on input (x, k), $\mathbb{A}$ makes queries to the oracle only among the last $h(k)$ steps of any computation path.

Because we will use different kinds of oracle machines, and the exponent notation for the relativization of a complexity class is difficult to customize, we will also use the (older) parenthesis notation: If C is a complexity class that is characterized by machines, we denote by $C(O)$ the class characterized by oracle machines of the same type as the ones characterizing C, with unrestricted access to the oracle O. Similarly, $C(O)_{bal}$ denotes the class defined by oracle machines with *balanced* access to the parameterized oracle, $C(O)_{para}$ denotes the class defined by oracle machines with *parameter-bounded* access to the oracle, and $C(O)_{tail}$ denotes the class defined by tail-nondeterministic oracle machines with the same restrictions as the machines that define C. The exponent notation is only used when the type of oracle access is the "natural" one for the type of machine being considered (so $\mathbf{A}[1]^O = \mathbf{A}[1](O)_{tail}$ and $\mathbf{W}[\mathbf{P}]^O = \mathbf{W}[\mathbf{P}](O)_{para}$). For **FPT** we always specify the type of oracle access.

Relativization results for tail-nondeterministic random access machines. We give an informal overview of the proof that $\mathbf{A}[1]^{p\text{-MC}(\Sigma_t[3])} = \mathbf{A}[t+1]$, to highlight the role played by the choice of the oracle and by the restrictions made to the tail-nondeterministic oracle machines (for a comparison with the proof that $\mathbf{NP}^{\Sigma_i \text{SAT}} = \mathbf{\Sigma}^P_{i+1}$, see [3], Section 5.5).

For the "$\supseteq$"-inclusion, we have that an $\mathbf{A}[1]$-machine with a p-MC($\Sigma_t[3]$)-oracle (which is complete for $\mathbf{A}[t]$) can first deterministically simulate the deterministic part of the computation of an $\mathbf{A}[t+1]$-machine on input (x, k). The oracle $\mathbf{A}[1]$-machine then enters the nondeterministic phase of its computation, and uses its own nondeterministic guesses to simulate the first block of existential guesses of the simulated machine (until a universal instruction is encountered). The computation of the $\mathbf{A}[t+1]$-machine from this point onward (which starts with a universal guess instruction and has $\leq t-1$ alternations) can be encoded as an instance $((\mathcal{A}, \phi), |\phi|)$ of p-MC($\Sigma_t[3]$) (see Remark 11), but the size of $\mathcal{A}$ depends on $|x|$. Therefore, $\mathcal{A}$ must (for the most part) be computed by the oracle $\mathbf{A}[1]$-machine and written to the oracle registers ahead of time, during the deterministic phase of the computation, with only the formula ϕ left to be computed during the nondeterministic phase. This is why it is necessary to allow tail-nondeterministic oracle machines access to their oracle registers throughout the entire computation.

For the reverse inclusion, we have that an $\mathbf{A}[t+1]$-machine can simulate an oracle $\mathbf{A}[1]$-machine on input (x, k), by first simulating the deterministic part of the computation deterministically, and then using $(t+1)$-alternating nondeterminism to simulate both the oracle $\mathbf{A}[1]$-machine's existential guesses, as well as all of the p-MC$(\Sigma_t[3])$-queries (this is accomplished in the same way as in the classical proof). In order to evaluate the queried instances, however, the $\mathbf{A}[t+1]$-machine's computation must be in its nondeterministic phase, so it is essential that:

- the simulated oracle machine can not make queries outside of the last $h(k)$ steps of its computation, for some computable function h;
- the size of the formulas in the queried instances is $\leq g(k)$, for some computable function g (balanced oracle access);
- the quantifier-free part of a formula can be evaluated efficiently (relational structures must be encoded in such a way that expressions involving relations can be evaluated by a RAM in time independent of the size of the relational structure; see Remark 8).

▶ **Theorem 13.** *For every $t \geq 1$, $\mathbf{A}[1]^{p\text{-MC}(\Sigma_t[3])} = \mathbf{A}[t+1]$.*

Since, for every $t \geq 1$, the problem used as an oracle in Theorem 13 is complete for $\mathbf{A}[t]$, it would be tempting to now state that $\mathbf{A}[1]^{\mathbf{A}[t]} = \mathbf{A}[t+1]$, because this would imply a "collapse theorem" for this hierarchy, namely that $\forall t \geq 1 : \mathbf{A}[t] = \mathbf{A}[t+1] \Rightarrow (\forall t' \geq t : \mathbf{A}[t] = \mathbf{A}[t'])$. Unfortunately, tail-nondeterminism appears to be too weak for such a collapse theorem to be proved in this fashion. In fact, it is not even certain whether $\mathbf{A}[1]^{\mathbf{FPT}} \subseteq \mathbf{A}[2]$: This is because an $\mathbf{A}[2]$-machine trying to simulate an $\mathbf{A}[1]$-machine that has oracle access to some non-trivial problem in $\mathbf{FPT}$, on some input (x, k), may have to enter the nondeterministic phase of its computation before it even knows the instance to be queried (the simulated machine may write a large instance to its oracle registers, and then nondeterministically make some changes to it before querying the oracle). The size of this instance may depend on $|x|$, and although it can be decided in fpt-time, it may not be possible to decide it in time $h(k)$, for some computable function h, even with 2-alternating nondeterminism. Thus, the property of p-MC$(\Sigma_t[3])$ that, with the right encoding, an instance $((\mathcal{A}, \phi), |\phi|)$ can be decided by a t-alternating tail-nondeterministic ARAM in time depending computably only on $|\phi|$, appears to have been crucial for our oracle characterization of the $\mathbf{A}$-Hierarchy.

The next theorem is the parameterized analogue of a famous classical result of Baker, Gill, and Solovay [4]. The construction of a parameterized oracle B relative to which $\mathbf{FPT}$ and $\mathbf{A}[1]$ differ, is done via diagonalization and uses similar ideas as the classical proof in [4], but with two noteworthy differences:

First, when diagonalizing against all $\mathbf{FPT}$-machines, we can not computably list all such machines, because the $f(k)$-term in their running times can be any computable function. We must therefore proceed more carefully with the construction in order to obtain an oracle which is computable.

Second, when running each RAM on larger and larger inputs for an increasing number of steps while constructing the oracle, we are free to increase *both* the size of the main part of the input *and* the parameter value. Having this additional dimension of the input works in our favor, and allows us to "kill" the $f(k)$-term in the running time of any $\mathbf{FPT}$-machine by increasing $|x|$ so that $|x| > f(k)$, at which point we can treat $f(k)|x|^c$ as a polynomial in $|x|$.

▶ **Theorem 14.** *There exist parameterized oracles A and B such that*

$$\mathbf{FPT}(A)_{tail} = \mathbf{A}[1]^A \quad \textit{and} \quad \mathbf{FPT}(B)_{tail} \subsetneq \mathbf{A}[1]^B \quad \textit{(and even } \mathbf{A}[1]^B \setminus \mathbf{FPT}(B) \neq \emptyset\textit{)}.$$

Relativization results for RAMs with parameter-bounded nondeterminism. For this machine model, we first need to define the analogue of the Polynomial Hierarchy.

▶ **Definition 15.** For each $t \geq 1$, let $\mathbf{\Sigma}_t^{[P]}$ be the class of parameterized problems that can be decided by a parameter-restricted t-alternating ARAM $\mathbb{A}$ such that, for some computable function h, on any input (x, k), $\mathbb{A}$ executes at most $h(k)$ nondeterministic instructions on any computation path. Furthermore, we define $\mathbf{W}[\mathbf{P}]\mathbf{H} := \bigcup_{t=1}^{\infty} \mathbf{\Sigma}_t^{[P]}$.

Clearly, $\mathbf{W}[\mathbf{P}] = \mathbf{\Sigma}_1^{[P]} \subseteq \mathbf{W}[\mathbf{P}]\mathbf{H} \subseteq \mathbf{AW}[\mathbf{P}]$. For $t \geq 2$, $\mathbf{\Sigma}_t^{[P]}$-complete problems can be obtained by modifying known $\mathbf{W}[\mathbf{P}]$- or $\mathbf{AW}[\mathbf{P}]$-complete problems appropriately (see [6, 11]).

We turn to the oracle characterization of this hierarchy. Since a $\mathbf{W}[\mathbf{P}]$-machine can compute fpt-reductions at any point in the computation, the choice of the complete problem given as an oracle is no longer important. Now the proof of the theorem proceeds in the same way as the characterization of **PH** in terms of oracle machines (see [3], Section 5.5), but note that for the "⊆"-inclusion, the restrictions on the oracle access are nevertheless essential: balanced access ensures that the $\mathbf{\Sigma}_{t+1}^{[P]}$-machine can nondeterministically decide the instances queried by the oracle machine, and parameter-bounded access ensures that the number of queries made by the oracle machine is not too large for a $\mathbf{\Sigma}_{t+1}^{[P]}$-machine to simulate.

▶ **Theorem 16.** *For each $t \geq 1$, we have $\mathbf{W}[\mathbf{P}]^{\mathbf{\Sigma}_t^{[P]}} = \mathbf{\Sigma}_{t+1}^{[P]}$.*

▶ **Corollary 17.** *For any $t, u \geq 1$, if $\mathbf{\Sigma}_t^{[P]} = \mathbf{\Sigma}_{t+u}^{[P]}$, then $\mathbf{W}[\mathbf{P}]\mathbf{H} = \mathbf{\Sigma}_t^{[P]}$.*

Finally, we have the oracle separation result for this machine model, as in [4]:

▶ **Theorem 18.** *There exist parameterized oracles A and B such that*

$$\mathbf{FPT}(A)_{para} = \mathbf{W}[\mathbf{P}]^A \quad \textit{and} \quad \mathbf{FPT}(B)_{para} \subsetneq \mathbf{W}[\mathbf{P}]^B \quad \textit{(and even } \mathbf{W}[\mathbf{P}]^B \setminus \mathbf{FPT}(B) \neq \emptyset\textit{).}$$

For the proof, it suffices to use the same two oracles as in the proof of Theorem 14.

4 Interactive proof systems for parameterized complexity classes

A classical interactive proof system consists of a verifier and a prover who exchange messages in order for the verifier to decide whether a given input is a 'yes'-instance of a problem. The verifier is a probabilistic TM, meaning that he can guess random bits, but his computation throughout the entire interaction is time-bounded polynomially in terms of the size of the input instance (and therefore so is the length of the messages he can send or receive). The prover is computationally all-powerful, but he only sees the input and the messages sent by the verifier (not the verifier's random bits), and his goal is to convince the verifier to accept. A proof system is said to decide a problem Q if every $x \in Q$ is accepted by the verifier with probability (over the verifier's random bits) $\geq 2/3$ for some prover, and every $x \notin Q$ is accepted by the verifier with probability $\leq 1/3$ for any prover (see [3], Chap. 8).

Here we make a slight change to this definition, in order to apply the concept to parameterized complexity classes, by letting the verifier be a probabilistic RAM (meaning that he can guess non-negative integers of bounded size in a single step), and allowing the messages between verifier and prover to be strings of non-negative integers of bounded size. This change does not affect the (classical) class **IP** (see Remark 2), but allows us to apply separate bounds to different aspects of the proof systems.

Arithmetization of first-order formulas with relational vocabularies. Before we can give interactive proof systems for parameterized complexity classes, we need to adapt the main technical tool used in such results, namely arithmetization. The main idea behind the original version of this technique is that a quantified Boolean formula can be replaced by a multivariate polynomial which coincides with the formula on all assignments of values to the non-quantified variables, if the Boolean truth values are identified with the elements of $GF(2)$ (in other words, the Boolean formula is encoded as a polynomial). Once this is accomplished, the polynomial can also be evaluated over some larger field, which is a key ingredient of the proof that **PSPACE** $\subseteq$ **IP** [15].

We wish to encode a pair $(\phi, \mathcal{A})$ as a polynomial, where ϕ is an FO formula, $\mathcal{A}$ is a relational structure with the same vocabulary as ϕ, and the universe A of $\mathcal{A}$ is $\{0, \ldots, n\}$, $n \in \mathbb{N} \setminus \{0\}$. The main obstacle here is that the atomic formulas in ϕ are not Boolean variables, but relational expressions of the form $Rx_1 \ldots x_l$, which evaluate to Boolean values whenever the variables are assigned values from A. We need a way to encode such a relational expression as a polynomial P_R that takes the values 0 or 1 whenever $x_1, \ldots, x_l \in A$, in accordance with the relation in $\mathcal{A}$ corresponding to R. To do this, we first choose a prime $q > n + 1$ and identify A with a subset of $\{0, \ldots, q - 1\}$. We then take P_R as the sum over all terms of the form $(1 - (X - a_1)^{q-1}) \cdot \ldots \cdot (1 - (X - a_l)^{q-1})$, where $(a_1, \ldots, a_l)$ is in the relation corresponding to R in $\mathcal{A}$, and argue via Fermat's Little Theorem that whenever P_R is evaluated over values from $GF(q)$, at most one such term is 1, the rest being 0, and that P_R therefore encodes the expression $Rx_1 \ldots x_l$. (See the full version of the paper for details.)

With arithmetization generalized in this way, we are now in a position to construct an IP similar to the one used in [16] to show that **PSPACE** $\subseteq$ **IP**, and prove the following:

▶ **Theorem 19.** *For every problem $Q \in$ **AW**[**SAT**], there is an interactive proof system deciding Q such that, for some computable functions f and h, and a polynomial p, on any input (x, k), the verifier runs in time $f(k)p(|x|)$, guesses at most $h(k)$ random numbers, and the interaction has at most $h(k)$ rounds.*

The IP in Theorem 19 has both the number of rounds and the number of random guesses made by the verifier bounded computably in terms of the parameter, but the length of the prover's messages and of the verifier's computations between rounds are "fpt-bounded". In order for an **AW**[$*$]-machine to simulate an interactive proof, it would presumably need to nondeterministically guess the prover's messages, as well as the random guesses made by the verifier, so the entire interaction would have to be simulated in the last $h(k)$ steps of the computation (due to tail-nondeterminism). In other words, the proof system would have to be such that the verifier only performs an fpt-bounded pre-computation, followed by an interaction that is entirely bounded in the parameter alone. We conjecture that the class of problems with such IPs, which we call $\mathbf{IP}^{tail}$, is precisely **AW**[$*$]. The evidence for this conjecture is that when the size of the FO formula is bounded in terms of the parameter, it seems that the IP from Theorem 19 can be improved so that at least the length of the prover's messages depends only on the parameter, by using only symbols for the polynomials representing the atomic relations, rather than expanding them into algebraic expressions. Getting the same bound for the verifier's computations between rounds is more challenging.

5 Conclusions

We have shown that, with some degree of effort, certain classical methods can be put to use in the parameterized setting, although some theorems only partially transfer over. The fact that different aspects of the computation of a RAM are bounded differently, and that some computational resources can be tail-restricted, ensures that the machine-based theory of parameterized intractability is by no means just “complexity theory with RAMs”.

One can now attempt to make some progress on the problem of separating matching levels of the **A**- and the **W**-Hierarchy by proving oracle separations when reasonable restrictions are placed on the oracle access of the respective machines. Another question is related to the fact that the implication $\mathbf{NP} \neq \mathbf{P} \Rightarrow \mathbf{A}[1] \neq \mathbf{FPT}$ is not known to hold: It would be interesting to show that this implication fails to hold relative to some oracle.

Acknowledgments The author is grateful to Yijia Chen and Sándor Kisfaludi-Bak for helpful discussions, to Martin Bottesch, Sándor Kisfaludi-Bak, and Ronald de Wolf for comments on a draft of this paper, and to two anonymous reviewers for their detailed suggestions on improvements to the version submitted to IPEC 2017.

References

1 S. Aaronson and A. Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1), 2009.

2 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.

3 S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge, 2009.

4 T. Baker, J. Gill, and R. Solovay. Relativizations of the P=?NP question. *SIAM J. Comput.*, 4(4):431–442, 1975.

5 Y. Chen and J. Flum. Machine characterizations of the classes of the W-hierarchy. *Proceedings of the 17th International Workshop on Computer Science Logic, Lecture Notes in Computer Science*, 2803:114–127, 2003.

6 Y. Chen, J. Flum, and M. Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. *Proceedings of the 18th IEEE Conference on Computational Complexity*, pages 13–29, 2003.

7 Y. Chen, J. Flum, and M. Grohe. Machine-based methods in parameterized complexity theory. *Theor. Comput. Sci.*, 339:167–199, 2005.

8 R.G. Downey and M.R. Fellows. *Parameterized Complexity.* Springer, Berlin, 1999.

9 R.G. Downey and M.R. Fellows. *Fundamentals of Parameterized Complexity.* Springer, 2013.

10 J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model checking. *SIAM J. Comput.*, 31(1):113–145, 2001.

11 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, Berlin, 2006.

12 L. Fortnow. A simple proof of Toda’s theorem. *Theory of Computing*, 5:135–140, 2009.

13 J.A. Montoya and M. M´uller. Parameterized random complexity. *Theory. Comput. Syst.*, 52:221–270, 2013.

14 C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

15 A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

16 A. Shen. IP = PSPACE: simplified proof. *J. ACM*, 39(4):878–880, 1992.

17 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.