Machine-based Methods in Parameterized Complexity Theory

Yijia Chen* Jörg Flum† Martin Grohe‡

May 12, 2004

Abstract

We give machine characterizations of most parameterized complexity classes, in particular, of W[P], of the classes of the W-hierarchy, and of the A-hierarchy. For example, we characterize W[P] as the class of all parameterized problems decidable by a nondeterministic fixed-parameter tractable algorithm whose use of nondeterminism is bounded in terms of the parameter. The machine characterizations suggest the introduction of a hierarchy W^{func} between the W and the A-hierarchy. We study the basic properties of this hierarchy.

1. Introduction

Parameterized complexity theory provides a framework for a fine-grain complexity analysis of algorithmic problems that are intractable in general. It has been used to analyze problems in various areas of computer science, for example, database theory [13, 15], artificial intelligence [12], and computational biology [1, 16]. The theory is built on a weakened notion of tractability called *fixed-parameter tractability*, which relaxes the classical notion of tractability, polynomial time computability, by admitting algorithms whose running time is exponential, but only in terms of some *parameter* of the problem instance that can be expected to be small in the typical applications.

A core structural parameterized complexity theory has been developed over the last 10–15 years (see [6]). Unfortunately, it has led to a bewildering variety of parameterized complexity classes, the most important of which are displayed in Figure 1. As the reader will have guessed, none of the inclusions is known to be strict. The smallest of the displayed classes, FPT, is the class of all fixed-parameter tractable problems. Of course there is also a huge variety of classical complexity classes, but their importance is somewhat limited by the predominant role of the class NP. In parameterized complexity, the classification of problems tends to be less clear cut. For example, for each of the classes $W[1]$, $W[2]$, and W[P] there are several natural complete problems which are parameterizations of classical NP-complete problems.

Not only is there a large number of (important) parameterized complexity classes, but unfortunately it is also not easy to understand these classes. The main reason for this may be seen in the fact that all the classes (except FPT) are defined in terms of complete problems, and no natural machine characterizations are known. This makes it hard to get a grasp on the classes, and it also frequently leads to confusion with respect to what notion of reduction is used to define the classes.[1] In this paper, we try to remedy this situation by giving machine characterizations of the parameterized complexity classes.

Our starting point is the class W[P], which is defined to be the class of all parameterized problems that are reducible to the *weighted satisfiability problem* for Boolean circuits. This problem asks whether a given circuit has a satisfying assignment of *weight* k, that is, a satisfying assignment in which precisely k inputs are set to TRUE. Here k is treated as the *parameter* of the problem. It is worth mentioning at this point that all the other "W-classes" in Figure 1 are defined similarly in terms of the weighted satisfiability problem, but for restricted classes of circuits. Our first theorem is a simple machine characterization of the class W[P], which generalizes an earlier characterization due to Cai, Chen, Downey and Fellows [2] of the

*Email: chen@zermelo.mathematik.uni-freiburg.de.

†Email: Joerg.Flum@math.uni-freiburg.de.

‡Email: grohe@informatik.hu-berlin.de

[1] Downey and Fellow's monograph [6] distinguishes between three types of Turing reductions, which also have corresponding notions of many-one reductions. In principle, there is a version of each of the complexity classes for each of the six forms of reduction.

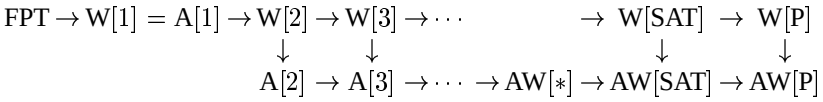

Figure 1. Parameterized complexity classes.

class of all problems in W[P] that are in NP when considered as classical problems. Intuitively, our result states that a problem is in W[P] if and only if it is decidable by a nondeterministic fixed-parameter tractable algorithm whose use of nondeterminism is bounded in terms of the parameter. A precise formulation of this result is that a problem is in W[P] if and only if it is decided in time $f(k) \cdot p(n)$ by a nondeterministic Turing machine that makes at most $f(k) \cdot \log n$ nondeterministic steps, for some computable function f and some polynomial p. Here k denotes the parameter and n the size of the input instance. While it has been noted before (see, for example, Chapter 17 of [6]) that there is a relation between limited nondeterminism and parameterized complexity theory, no such simple and precise equivalence was known. As a by-product of this result, we get a somewhat surprising machine characterization of the class $\mathrm{W}[1] = \mathrm{A}[1]$: A problem is in $\mathrm{W}[1]$ if and only if it is decidable by a nondeterministic fixed-parameter tractable algorithm that does its nondeterministic steps only among the last steps of the computation.

Nondeterministic random access machines turn out to be the appropriate machine model in order to make precise what we mean by "among the last steps". In their nondeterministic steps these machines are able to guess natural numbers $\leq f(k) \cdot p(n)$, where f is a computable function and p a polynomial. The corresponding alternating random access machines characterize the classes of the A-hierarchy.

It is known that the model-checking problems MC(Σ_t) and MC($\Sigma_{t,1}$) are complete for $\mathrm{A}[t]$ and $\mathrm{W}[t]$, respectively. Here, $\Sigma_{t,1}$ denotes the class of Σ_t-formulas in relational vocabularies, where all blocks of quantifiers, besides the first one, just consist of a single quantifier. The corresponding restriction for the length of the blocks of alternating random access machines yield machine characterizations of the classes of a hierarchy, which we denote by $\mathrm{W}^{\mathrm{func}}$. We have $\mathrm{W}[t] \subseteq \mathrm{W}^{\mathrm{func}}[t] \subseteq \mathrm{A}[t]$, but we do not know, if any of the inclusions can be replaced by an equality for $t \geq 2$. We show that the model-checking problem for $\Sigma_{t,1}$-formulas in vocabularies *with* function symbols is complete for $\mathrm{W}^{\mathrm{func}}[t]$.

To obtain machine characterizations of the classes of the W-hierarchy we have to restrict the access of alternating random access machines to the guessed numbers; in a certain sense they only have access to properties of the objects denoted by the numbers but not to the numbers themselves.

Extended abstracts containing parts of the results appeared as [3, 4].

2. Parameterized Complexity Theory

We recall the notions of parameterized problem, of fixed-parameter tractability, and of fpt-reduction.

A *parameterized problem* is a set $Q \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. If $(x, k) \in \Sigma^* \times \mathbb{N}$ is an instance of a parameterized problem, we refer to x as the *input* and to k as the *parameter*. Usually, we denote the parameter by k and the length of the input string x by n. Parameters are encoded in unary, although, in general, this is inessential.

Definition 1. A parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is *fixed-parameter tractable*, if there is a computable function $f : \mathbb{N} \to \mathbb{N}$, a polynomial p, and an algorithm that, given a pair $(x, k) \in \Sigma^* \times \mathbb{N}$, decides if $(x, k) \in Q$ in at most $f(k) \cdot p(n)$ steps.

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

Occasionally we use the term fpt-*algorithm* to refer to an algorithm that takes as input pairs $(x, k) \in \Sigma^* \times \mathbb{N}$ and has a running time bounded by $f(k) \cdot p(n)$ for some computable function $f : \mathbb{N} \to \mathbb{N}$ and polynomial p. Thus a parameterized problem is in FPT, if it can be decided by an fpt-algorithm. However, we use the term fpt-algorithm mostly when referring to algorithms computing mappings.

To compare the complexity of problems that are not fixed-parameter tractable, we need an appropriate notion of parameterized reduction:

Definition 2. An fpt-*reduction* from the parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ to the parameterized problem $Q' \subseteq (\Sigma')^* \times \mathbb{N}$ is a mapping $R : \Sigma^* \times \mathbb{N} \to (\Sigma')^* \times \mathbb{N}$ such that:

(1) For all $(x, k) \in \Sigma^* \times \mathbb{N}$: $(x, k) \in Q \iff R(x, k) \in Q'$.

(2) There exists a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$, say with $R(x, k) = (x', k')$, we have $k' \leq g(k)$.

(3) R can be computed by an fpt-algorithm.

We write $Q \leq^{\text{fpt}} Q'$ if there is an fpt-reduction from Q to Q'. We let

$$[Q]^{\text{fpt}} \quad := \quad \{Q' \mid Q' \leq^{\text{fpt}} Q\}.$$

Often we introduce parameterized problems in the form we do here with the *parameterized clique problem* p-CLIQUE:

p-CLIQUE	
Input:	A graph $\mathcal{G}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{G}$ has a clique of size k.

3. The class W[P]

One of the most natural NP-complete problems is the *circuit satisfiability problem*, the problem to decide if a given circuit can be satisfied by some assignment. A circuit $\mathcal{C}$ is *k-satisfiable* (where $k \in \mathbb{N}$), if there is an assignment for the set of input gates of $\mathcal{C}$ of weight k satisfying $\mathcal{C}$, where the *weight* of an assignment is the number of input gates set to TRUE. The *weighted circuit satisfiability problem* p-WSATCIRCUIT is the following parameterized problem:

p-WSATCIRCUIT	
Input:	A circuit $\mathcal{C}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{C}$ is k-satisfiable.

Now, W[P] is the class of all parameterized problems that are fpt-reducible to p-WSATCIRCUIT, that is,

$$\text{W[P]} \quad := \quad [p\text{-WSATCIRCUIT}]^{\text{fpt}}.$$

Cai et al. [2] gave a machine characterization of all problems in W[P] which are in NP when considered as classical problems.

Theorem 3 (Cai et al. [2]). *Let Q be a parameterized problem, which is in* NP *as a classical problem. Then $Q \in$ W[P] if and only if there is a nondeterministic Turing machine M deciding Q such that M on input (x, k) performs at most $p(|x| + k)$ steps and at most $f(k) \cdot \log n$ nondeterministic steps (for some computable f and polynomial p).*

Clearly, there are W[P] problems that do not lie in NP. In this section, we generalize Theorem 3 to a machine characterization that covers all of W[P].

We use standard random access machines (RAMs) as described in [14]. Deviating from [14], we assume for simplicity, that the registers contain natural numbers (and not integers). The arithmetic operations are addition, (modified) subtraction, and division by two (rounded off), and we use a uniform cost measure. For details, we refer the reader to Section 2.6 of [14]. If the machine stops, it *accepts* its input, if the content of register 0 (the *accumulator*) is 0; otherwise, it *rejects*.

Our model is non-standard when it comes to nondeterminism. Instead of just allowing our machines to nondeterministically choose one bit, or an instruction of the program to be executed next, we allow them to nondeterministically choose a natural number. Of course this is problematic, because if the machine can really "guess" arbitrary numbers, computations can no longer be described by finitely branching trees,

and nondeterministic machines can no longer be simulated by deterministic ones. To avoid the kind of problems resulting from this, we decided that a "bounded" version of this unlimited nondeterminism is most appropriate for our purposes. Therefore, we define a *nondeterministic* RAM, or NRAM, to be a RAM with an additional instruction "GUESS" whose semantics is: Guess a natural number less than or equal to the number stored in the accumulator and store it in the accumulator. Acceptance of an input by an NRAM program is defined as usually for nondeterministic machines. Steps of a computation of an NRAM that execute a GUESS instruction are called *nondeterministic steps*.

While this form of nondeterminism may seem unnatural at first sight, we would like to argue that it is very natural in many typical "applications" of nondeterminism. For example, a nondeterministic algorithm for finding a clique in a graph guesses a sequence of vertices of the graph and then verifies that these vertices indeed form a clique. Such an algorithm is much easier described on a machine that can guess the numbers representing the vertices of a graph at once, rather than guessing their bits. In any case, we believe that our results justify our choice of model. For a further discussion of this issue we refer the reader to Remark 9.

Definition 4. A program $\mathbb{P}$ for an NRAM is an nfpt-*program*, if there is a computable function f and a polynomial p such that for every input (x, k) with $|x| = n$ the program $\mathbb{P}$ on every run

(1) performs at most $f(k) \cdot p(n)$ steps;

(2) performs at most $f(k)$ nondeterministic steps;

(3) uses at most the first $f(k) \cdot p(n)$ registers;

(4) contains numbers $\leq f(k) \cdot p(n)$ in all registers at every point of the computation.

By standard arguments one gets:

Lemma 5. *Let Q be a parameterized problem. The following are equivalent:*

(1) There is an nfpt-*program for an* NRAM *deciding Q.*

(2) There is a nondeterministic Turing machine M deciding Q such that M on input (x, k) performs at most $g(k) \cdot q(n)$ steps and at most $g(k) \cdot \log n$ nondeterministic steps (for some computable function g and polynomial q; recall that n denotes the length $|x|$ of x).

(3) There is a nondeterministic Turing machine M accepting Q. Moreover, every accepting run of M on input (x, k) has length at most $g(k) \cdot q(n)$ and has the nondeterministic steps among the first $g(k) \cdot \log n$ ones (for some computable function g and polynomial q).

Theorem 6. *Let Q be a parameterized problem. Then $Q \in$* W[P] *if and only if there is an* nfpt-*program for an* NRAM *deciding Q.*

Proof: Assume first that $Q \in$ W[P]. Then, by the definition of W[P], $Q \leq^{\text{fpt}} p$-WSATCIRCUIT. Hence there are computable functions f and g, a polynomial p, and an algorithm $\mathbb{A}$ assigning to every (x, k), in time $\leq f(k) \cdot p(n)$, a circuit $\mathcal{C}_{x,k}$ and a natural number $k' = k'(x, k) \leq g(k)$ such that

$$Qxk \quad \Longleftrightarrow \quad \mathcal{C}_{x,k} \text{ has a satisfying assignment of weight } k'.$$

Thus, we can assume that the nodes of the circuit $\mathcal{C}_{x,k}$ are (labelled by) natural numbers $\leq f(k) \cdot p(n)$. The claimed nfpt-program $\mathbb{P}$ on input (x, k) proceeds as follows:

1. It computes $\mathcal{C}_{x,k}$ and k';

2. It guesses the k' (labels of) input nodes to be set to TRUE;

3. It evaluates the circuit $\mathcal{C}_{x,k}$ and accepts (x, k) if the circuit outputs TRUE.

(When carrying out line 1, $\mathbb{P}$ simulates the algorithm $\mathbb{A}$ step by step and after each step increases a fixed register, say register i_0 by "1". Line 2 can be realized by k' times copying the content of register i_0 into the accumulator, invoking the instruction GUESS, and storing the guesses appropriately.) Clearly, the

number of steps that $\mathbb{P}$ performs can be bounded by $h(k) \cdot q(n)$ (for some computable function h and some polynomial q) and the number of nondeterministic steps is k' ($\leq g(k)$).

For the converse direction suppose that Q is decided by an nfpt-program $\mathbb{P}$. By the previous lemma, there is a computable function f, a polynomial p, and a nondeterministic Turing machine M accepting Q such that for $(x,k) \in Q$ every run of M accepts (x,k) in at most $f(k) \cdot p(n)$ steps and such that the nondeterministic steps are among the $f(k) \cdot \log n$ first ones. Without loss of generality, we may suppose that on every input, M first carries out the nondeterministic steps and that they altogether consist in appending to the input (x,k) a 0–1 string of length at most $f(k) \cdot \log n$.

The deterministic part of the computation of M can be simulated by a circuit $\mathcal{C}_{x,k}$ in the standard way (e.g., compare the proof of Theorem 8.1 in [14]) such that

$$M \text{ accepts } (x,k) \iff \mathcal{C}_{x,k} \text{ has a satisfying assignment.} \tag{1}$$

$\mathcal{C}_{x,k}$ has size $\leq g(k) \cdot q(n)$ for some computable function g and some polynomial q. It has $f(k) \cdot \log n$ input nodes corresponding to the 0–1 string chosen in the nondeterministic part of the computation of M (if more bits are required by the deterministic part of the computation of M, the circuit $\mathcal{C}_{x,k}$ will not accept the corresponding assignment).

We think of the $f(k) \cdot \log n$ input nodes of $\mathcal{C}_{x,k}$ as being arranged in $f(k)$ blocks of $\log n$ nodes. Let us obtain the circuit $\mathcal{D}_{x,k}$ by adding $f(k)$ blocks of n new input nodes to $\mathcal{C}_{x,k}$ and by ensuring that at most one input node of each block can be set to TRUE (in a satisfying assignment of $\mathcal{D}_{x,k}$). Moreover, we wire the new input nodes with the old input nodes (i.e., the input nodes of $\mathcal{C}_{x,k}$) in such a way that if the jth input node of the ith block of $\mathcal{D}_{x,k}$ is set to TRUE, then exactly those old input nodes of the ith block of $\mathcal{C}_{x,k}$, which correspond to positions of the binary representation of j carrying a 1, are set to TRUE. Then

$$\mathcal{C}_{x,k} \text{ has a satisfying assignment} \iff \mathcal{D}_{x,k} \text{ has a satisfying assignment of weight } f(k).$$

Altogether, we have shown that $Q \leq^{\text{fpt}} p\text{-WSATCIRCUIT}$, i.e. $Q \in \text{W[P]}$. □

Remark 7. Some of the arguments in the second half of the previous proof have been used by Downey and Fellows [6] in a similar context. Specifically, the arguments leading to (1) and hence, to the equivalence

$$(x,k) \in Q \iff \mathcal{C}_{x,k} \text{ has a satisfying assignment}$$

show that $Q \leq^{\text{fpt}}$ SHORT CIRCUIT SATISFIABILITY (cf. [6]). The transition from $\mathcal{C}_{x,k}$ to $\mathcal{D}_{x,k}$ duplicates the proof of [6] showing that W[P] contains SHORT CIRCUIT SATISFIABILITY; there, the method is called the $k \cdot \log n$ trick.

By Lemma 5 and Theorem 6 we can strengthen Theorem 3 by:

Corollary 8. *Let Q be a parameterized problem. Then $Q \in$ W[P] if and only if there is a nondeterministic Turing machine M deciding Q such that M on input (x,k) performs at most $g(k) \cdot q(n)$ steps and at most $g(k) \cdot \log n$ nondeterministic steps (for some computable function g and some polynomial q).*

Remark 9. The previous corollary shows that if we define nondeterministic RAMs by allowing the machines to guess only one bit per nondeterministic step instead of an arbitrary number, then Theorem 6 remains true if we allow an fpt-program to perform $f(k) \cdot \log n$ nondeterministic steps (cf. clause (2) in Definition 4).

The reason that we chose our non-standard definition of nondeterministic RAMs is that it also gives us a nice machine description of the class W[1] (see Theorem 15).

As a further corollary, we establish a connection between the collapse of W[P] with FPT and the existence of subexponential time algorithm of NP problems with bounded binary nondeterminism. The reader should compare our result with Corollary 17.3 in [6]. We could not verify the claim of this corollary, in fact there seems to be a gap in the proof.

We denote by $\text{NP}^*[f(k)]$ the class of problems $Q \subseteq \Sigma^* \times \mathbb{N}$ such that Qxk is solvable by a nondeterministic polynomial time (in $|x| + k$) algorithm that uses at most $f(k)$ bits of nondeterminism.

Similarly, SUBEXPTIME$^*[f(k)]$ denotes the class of problems $Q \subseteq \Sigma^* \times \mathbb{N}$ such that Qxk is solvable by a deterministic algorithm in time $p(|x| + k) \cdot 2^{g(k)}$ for some polynomial p and computable function $g \in o^{\text{eff}}(f)$. Here, $g \in o^{\text{eff}}(f)$ means that $g \in o(f)$ holds effectively, that is, there is a computable function h such that for all $\ell \geq 1$ and $m \geq h(\ell)$, we have $g(m)/f(m) \leq 1/\ell$.

Corollary 10. *The following are equivalent:*

(1) W[P] = FPT.

(2) For every PTIME*-function* f*:* NP$^*[f(k)] \subseteq$ SUBEXPTIME$^*[f(k)]$.

(3) NP$^*[\text{id}(k)] \subseteq$ SUBEXPTIME$^*[\text{id}(k)]$ *(where* id *denotes the identity function on* $\mathbb{N}$*).*

Proof: $(3) \Rightarrow (1)$: Assume (3). It suffices to show that p-WSATCIRCUIT $\in$ FPT. Define the following classical problem Q:

Q	
Input:	A circuit $\mathcal{C}$ and $k \in \mathbb{N}$.
Problem:	Decide if $\mathcal{C}$ is $\frac{k}{\log \|\mathcal{C}\|}$-satisfiable.

Here, $\|\mathcal{C}\|$ denotes the length of a string encoding the circuit $\mathcal{C}$ in a reasonable way. Clearly, we have $Q \in$ NP$^*[\text{id}(k)]$ and hence by assumption (3), $Q \in$ SUBEXPTIME$^*[\text{id}(k)]$. Moreover, for any instance $(\mathcal{C}, k)$ of p-WSATCIRCUIT,

$$(\mathcal{C}, k) \in p\text{-WSATCIRCUIT} \iff (\mathcal{C}, k \cdot \log \|\mathcal{C}\|) \in Q.$$

Therefore, for some polynomials p and q, p-WSATCIRCUIT can be decided in time

$$q(\|\mathcal{C}\| + k) + p(\|\mathcal{C}\| + k \cdot \log \|\mathcal{C}\|) \cdot 2^{g(k \cdot \log \|\mathcal{C}\|)}$$

for some computable $g \in o^{\text{eff}}(\text{id})$. This implies that p-WSATCIRCUIT eventually is in PTIME and hence, p-WSATCIRCUIT $\in$ FPT.

The implication $(2) \Rightarrow (3)$ being trivial, we turn to a proof of $(1) \Rightarrow (2)$: Assume that W[P] = FPT. Let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a problem in NP$^*[f(k)]$ for some PTIME-function f. Choose an algorithm $\mathbb{A}$ witnessing $Q \in$ NP$^*[f(k)]$. We consider the following parameterization Q_p of Q:

Q_p	
Input:	$m \in \mathbb{N}$ in unary and an instance (x, k) of Q.
Parameter:	$\ell \in \mathbb{N}$.
Problem:	Decide if $f(k) \leq \ell \cdot \log m$ and $(x, k) \in Q$.

The following nfpt-program for an NRAM decides Q_p. The program

(1) checks whether $f(k) \leq \ell \cdot \log m$;

(2) guesses natural numbers $m_1, \ldots, m_\ell \leq m$;

(3) calculates the binary expansion of every m_i, altogether obtaining $\ell \cdot \log m$ $(\geq f(k))$ bits;

(4) using the first $f(k)$ bits in the nondeterministic steps, simulates the computation of $\mathbb{A}$ on input (x, k) and outputs the corresponding answer.

By our assumption W[P] = FPT, we have $Q_p \in$ FPT. Therefore, there is an algorithm $\mathbb{A}_1$ that decides $Q_p mxk\ell$ in time $g(\ell) \cdot (m + |x| + k)^c$ for some computable g and $c \in \mathbb{N}$. By an argument, standard in complexity theory, we can assume that g is monotone and that g^{-1} is computable in polynomial time.

We present an algorithm $\mathbb{A}_2$ witnessing that $Q \in$ SUBEXPTIME$^*[f(k)]$. Given (x, k), this algorithm first computes

$$\ell := g^{-1}(k) \quad \text{and} \quad m := 2^{f(k)/\ell}$$

in polynomial time and in time $2^{o^{\text{eff}}(f(k))}$, respectively. Then, $f(k) \leq \ell \cdot \log m$. Now, $\mathbb{A}_2$ uses the algorithm $\mathbb{A}_1$ to decide, if $Q_p mxk\ell$ and hence, if Qxk. This requires time

$$\begin{aligned} g(\ell) \cdot (m + |x| + k)^c &\leq g(\ell) \cdot m^c \cdot (|x| + k)^c \\ &\leq k \cdot 2^{(c \cdot f(k))/\ell} \cdot (|x| + k)^c \\ &\leq (|x| + k)^{c+1} \cdot 2^{o^{\text{eff}}(f(k))}. \end{aligned}$$

Altogether, we get $Q \in \text{SUBEXPTIME}^*[f(k)]$. □

4. The class W[1]

In this section we present a machine characterization of the class W[1] (= A[1]). Similar to W[P], the classes W[1], W[2], ... of the W-hierarchy were also defined by weighted satisfiability problems for classes of circuits or propositional formulas. In particular, the weighted satisfiability problem for formulas in 3CNF is W[1]-complete. We will introduce the classes of the W-hierarchy by means of model-checking problems for fragments of first-order logic, since they are more appropriate for a discussion of the machine characterizations of the classes W[2], W[3], ... that we present in Section 7.

4.1. First-Order Logic and Model-Checking Problems. A *relational vocabulary* τ is a finite set of relation symbols. Each relation symbol has an *arity*. The arity of τ is the maximum of the arities of the symbols in τ. A *structure* $\mathcal{A}$ of vocabulary τ, or τ-structure, consists of a set A called the *universe*, an interpretation $R^{\mathcal{A}} \subseteq A^r$ of each r-ary relation symbol $R \in \tau$. We synonymously write $\bar{a} \in R^{\mathcal{A}}$ or $R^{\mathcal{A}}\bar{a}$ to denote that the tuple $\bar{a} \in A^{\text{arity}(R)}$ belongs to the relation $R^{\mathcal{A}}$. *We only consider structures whose universe is finite*. The *size* of a τ-structure $\mathcal{A}$ is the number

$$\|\mathcal{A}\| \quad := \quad |\tau| + |A| + \sum_{R \in \tau} \text{arity}(R) \cdot |R^{\mathcal{A}}|,$$

which is the size of the *list representation* of $\mathcal{A}$ (cf. [10]).

Example 11. We view a *directed graph* as a structure $\mathcal{G} = (G, E^{\mathcal{G}})$, whose vocabulary consists of one binary relation symbol E. $\mathcal{G}$ is an (undirected) *graph*, if $E^{\mathcal{G}}$ is irreflexive and symmetric. GRAPH denotes the class of all graphs.

The class of all first-order formulas is denoted by FO. They are built up from atomic formulas using the usual boolean connectives and existential and universal quantification. Recall that *atomic formulas* are formulas of the form $x = y$ or $Rx_1 \ldots x_r$, where $x, y, x_1, \ldots, x_r$ are variables and R is an r-ary relation symbol. For $t \geq 1$, Σ_t denotes the class of all first-order formulas of the form

$$\exists x_{11} \ldots \exists x_{1k_1} \forall x_{21} \ldots \forall x_{2k_2} \ \ldots \ Qx_{t1} \ldots Qx_{tk_t} \ \psi,$$

where $Q = \forall$ if t is even and $Q = \exists$ otherwise, and where ψ is quantifier-free.

If $\mathcal{A}$ is a structure, $a_1, \ldots, a_m$ are elements of the universe A of $\mathcal{A}$, and $\varphi(x_1, \ldots, x_m)$ is a first-order formula whose free variables are among $x_1, \ldots, x_m$, then we write $\mathcal{A} \models \varphi(a_1, \ldots, a_m)$ to denote that $\mathcal{A}$ satisfies φ if the variables $x_1, \ldots, x_m$ are interpreted by $a_1, \ldots, a_m$, respectively. If φ is a *sentence*, i.e., a formula without free variables, then we write $\mathcal{A} \models \varphi$ to denote that $\mathcal{A}$ satisfies φ.

If Φ is a class of formulas, then $\Phi[\tau]$ denotes the class of all formulas of vocabulary τ in Φ and $\Phi[r]$, for $r \in \mathbb{N}$, the class of all formulas in Φ whose vocabulary has arity $\leq r$.

For a class D of structures and a class Φ of formulas, whose membership is PTIME-decidable, the *model-checking problem for Φ on D*, denoted by p-MC(D, Φ), is the problem of deciding whether a given structure $\mathcal{A} \in D$ satisfies a given sentence $\varphi \in \Phi$ parameterized by the length of φ, denoted by $|\varphi|$,

p-MC(D, Φ)	
Input:	$\mathcal{A} \in D$, a sentence $\varphi \in \Phi$.
Parameter:	$\|\varphi\|$.
Problem:	Decide if $\mathcal{A} \models \varphi$.

If D is the class of all finite structures, we also write p-MC(Φ) for p-MC(D, Φ).

Example 12. Note that a graph $\mathcal{G}$ has a clique of size k if and only if

$$\mathcal{G} \models \exists x_1 \ldots \exists x_k \bigwedge_{1 \leq i < j \leq k} Ex_i x_j.$$

Therefore, p-CLIQUE $\leq^{\text{fpt}}$ p-MC(GRAPH, Σ_1) $\leq^{\text{fpt}}$ p-MC(Σ_1).

The following definition of W[1] is the most appropriate for our purposes (its equivalence to the original definition was shown in [11]).

Definition 13.

$$\text{W}[1] := [p\text{-MC}(\Sigma_1)]^{\text{fpt}}.$$

It is known that p-CLIQUE is complete for W[1], so the following results are immediate by Example 12.

Theorem 14 (Downey, Fellows, Regan [7], Flum, Grohe [10, 11]).

(1) $\text{W}[1] = [p\text{-MC}(\text{GRAPH}, \Sigma_1)]^{\text{fpt}}$.

(2) For every relational vocabulary τ of arity ≥ 2,

$$\text{W}[1] = [p\text{-MC}(\Sigma_1[\tau])]^{\text{fpt}}.$$

The machine characterization of the class W[1] reads as follows:

Theorem 15. *Let Q be a parameterized problem. Then $Q \in$ W[1] if and only if there is a computable function h and an* nfpt*-program $\mathbb{P}$ for an* NRAM *deciding Q such that for every run of $\mathbb{P}$, all nondeterministic steps are among the last $h(k)$ steps of the computation, where k is the parameter.*

To express properties in first-order logic in a more readable fashion, it is sometimes advantageous to enlarge the vocabularies by constant symbols. Recall that in a given structure $\mathcal{A}$, a constant symbol d is interpreted by an element $d^{\mathcal{A}} \in A$. We will tacitly make use of the following lemma in the next proof:

Lemma 16. *There is a polynomial time algorithm that, given a τ-structure $\mathcal{A}$ and a τ-sentence $\varphi \in \Sigma_1$, where the vocabulary τ may contain constant symbols, computes a structure $\mathcal{A}'$ and a sentence $\varphi' \in \Sigma_1$ such that*

- *$\mathcal{A} \models \varphi \iff \mathcal{A}' \models \varphi'$;*
- *the vocabulary of $\mathcal{A}'$ and φ' is obtained from τ by replacing each constant symbol by a new unary relation symbol;*
- *φ' only depends on φ.*

Proof of Theorem 15: First assume that $Q \in$ W[1]. By Theorem 14,

$$Q \leq^{\text{fpt}} p\text{-MC}(\text{GRAPH}, \Sigma_1).$$

Hence, there is an fpt-algorithm assigning to every instance (x, k) of Q a graph $\mathcal{G} = \mathcal{G}_{x,k}$ and a sentence $\varphi = \varphi_{x,k} \in \Sigma_1$, say,

$$\varphi = \exists x_1 \ldots \exists x_{k'} \psi,$$

with $|\varphi| \leq g(k)$ for a computable function g, and with a quantifier-free ψ, such that

$$Qxk \iff \mathcal{G} \models \varphi.$$

Without loss of generality, the universe G of $\mathcal{G}$ is an initial segment of the natural numbers.

The following nfpt-program decides whether $(x, k) \in Q$:

1. It computes the graph $\mathcal{G}$ and stores the *adjacency matrix* of $\mathcal{G}$, i.e., for $u, v \in G$ a certain register (whose address is easily calculable from u, v) contains the information whether there is an edge between u and v.

2. It computes φ.

3. It checks whether $\mathcal{G} \models \varphi$.

To carry out point 3, the program, guesses the values of the quantified variables $x_1, \ldots, x_{k'}$. Then, it checks if the quantifier-free part ψ is satisfied by this assignment. Since we stored the adjacency matrix of $\mathcal{G}$, the number of steps needed for point 3 can be bounded by $h(k)$ for some computable h. Hence, all nondeterministic steps are among the last $h(k)$ steps of the computation.

Assume now that the nfpt-program $\mathbb{P} = (\pi_1, \ldots, \pi_m)$ for an NRAM decides Q and that for some computable function h, on every run of $\mathbb{P}$ on input (x, k) the nondeterministic steps are among the last $h(k)$ ones. Choose a computable function f and a polynomial p for $\mathbb{P}$ according to the definition of nfpt-program. The set of *instruction numbers* of $\mathbb{P}$ is $\{1, \ldots, m\}$, more precisely, π_c is the instruction of $\mathbb{P}$ with instruction number c.

We show that $Q \leq^{\text{fpt}} p\text{-MC}(\Sigma_1)$. That is to say, for any instance (x, k) of Q, we will construct a structure $\mathcal{A} = \mathcal{A}_{x,k}$ and a Σ_1-sentence $\varphi = \varphi_{x,k}$ such that

$$\begin{aligned} Qxk &\iff \mathbb{P} \text{ accepts } (x, k) \\ &\iff \mathcal{A} \models \varphi. \end{aligned}$$

Let $\tau := \{\leq, R_+, R_-, R_{\text{div}}, \text{Reg}, 0, 1, \ldots, m, d_1, \ldots, d_s\}$ with relation symbols $\leq$ (binary), R_+, R_- (ternary), R_{div}, Reg (binary), and with the constant symbols $0, 1, \ldots, m, d_1, \ldots, d_s$. The τ-structure $\mathcal{A}$ has universe

$$A := \{0, 1, \ldots, f(k) \cdot p(n)\}.$$

(Without loss of generality, we assume $f(k) \cdot p(n) \geq \max\{m, d_1, \ldots, d_s\}$.) Furthermore

> $\leq^{\mathcal{A}}$ is the natural ordering on A;
> $R_+^{\mathcal{A}}, R_-^{\mathcal{A}}$, and $R_{\text{div}}^{\mathcal{A}}$ are the relations representing addition, subtraction, and division by two, respectively; for example, for $a_1, a_2, a_3 \in A$, we have ($R_+^{\mathcal{A}} a_1 a_2 a_3 \iff a_1 + a_2 = a_3$);
> for $a, b \in A$, $\text{Reg}^{\mathcal{A}} ab \iff b$ is the value of the ath register immediately before the first nondeterministic step is carried out;
> $0^{\mathcal{A}} = 0$, $1^{\mathcal{A}} = 1$, ..., $m^{\mathcal{A}} = m$; and $d_1^{\mathcal{A}}, \ldots, d_s^{\mathcal{A}}$ are the natural numbers occurring in the program $\mathbb{P}$ either as operands or as instruction numbers.

Clearly, $\mathcal{A}$ can be computed in the time allowed by an fpt-reduction.

Note that $\mathbb{P}$ (as any program for an NRAM) in each step of its computation changes the value of at most one register. In order to have a uniform terminology, we say that register 0 is changed to its actual value, if no register is updated.

Let $\bar{v}$ be the sequence of variables $x_1 y_1 z_1 \ldots x_{h(k)} y_{h(k)} z_{h(k)}$. For $\ell = 0, \ldots, h(k)$ we introduce formulas

$$\varphi_\ell(\bar{v}_\ell, x_{\ell+1}) \quad \text{and} \quad \psi_\ell(\bar{v}_\ell, y, z)$$

with $\bar{v}_\ell = x_1 y_1 z_1 \ldots x_\ell y_\ell z_\ell$ and with the meaning in $\mathcal{A}$:

> If on the nondeterministic part (the part beginning with the first nondeterministic step) of its run on instance (x, k), the program $\mathbb{P}$, so far, has carried out the instructions with numbers $x_1, \ldots, x_\ell$ thereby changing the content of register y_1 to z_1, ..., the content of register y_ℓ to z_ℓ, then the next instruction number is $x_{\ell+1}$,

and

if on the nondeterministic part of its run on instance (x,k), the program $\mathbb{P}$, so far, has carried out the instructions with numbers $x_1,\dots,x_\ell$ thereby changing the content of register y_1 to z_1, …, the content of register y_ℓ to z_ℓ, then the content of register y is z,

respectively. Let c_1 be the instruction number of the first nondeterministic step of $\mathbb{P}$ on (x,k), and c_0 the instruction number of the STOP instruction (without loss of generality, we assume that there is only one STOP instruction in $\mathbb{P}$). Recalling that $\mathbb{P}$ accepts its input, if the value of register 0 is 0 when stopping, we see that

$$\begin{aligned} & \mathbb{P} \text{ accepts } (x,k) \\ \iff \quad & \mathcal{A} \models \exists \bar{v} \bigvee_{0\le j<h(k)} \Big(x_{j+1} = c_0 \wedge \psi_j(\bar{v}_j,0,0) \wedge \bigwedge_{0\le \ell\le j} \varphi_\ell(\bar{v}_\ell, x_{\ell+1}) \Big), \end{aligned}$$

which gives the desired reduction from Q to p-MC(Σ_1).

The formulas ψ_ℓ and φ_ℓ are defined by induction on i:

$$\begin{aligned} \psi_0(y,z) &:= \operatorname{Reg} yz, \\ \psi_{\ell+1}(\bar{v}_{\ell+1},y,z) &:= (y = y_{\ell+1} \to z = z_{\ell+1}) \wedge (y \neq y_{\ell+1} \to \psi_\ell(\bar{v}_\ell,y,z)), \\ & \quad \text{“if the register } y \text{ is updated in the } (\ell+1)\text{th step, then its content is } z_{\ell+1}, \\ & \quad \text{otherwise it is the same as after the } \ell\text{th step”}. \end{aligned}$$

We turn to the definition of φ_ℓ:

$$\begin{aligned} & & \varphi_0(x_1) &:= x_1 = c_1, \\ \text{for } \ell \ge 1 \quad & & \varphi_\ell(\bar{v}_\ell, x_{i+1}) &:= \textstyle\bigvee_{1\le c\le m} (x_\ell = c \wedge \varphi^c_\ell(\bar{v}_\ell, x_{\ell+1})), \end{aligned}$$

where each φ^c_i depends on the instruction π_c. Say $\pi_c = \text{READ} \uparrow u$ (i.e., “store the number in register v in register 0, where v is the content of the uth register”). Let d be a constant symbol with $d^{\mathcal{A}} = u$. Then, we set

$$\varphi^c_\ell(\bar{v}_\ell, x_{\ell+1}) := \exists z(\psi_{\ell-1}(\bar{v}_\ell - 1, d, z) \wedge \psi_{\ell-1}(\bar{v}_{\ell-1}, z, z_\ell) \wedge y_\ell = 0) \wedge R_+ x_\ell \, 1 \, x_{\ell+1}.$$

If $\pi_c = \text{JZERO } c'$ (i.e., $\pi_c =$ “if the content of register 0 is 0, then jump to the instruction $\pi_{c'}$”), then

$$\begin{aligned} \varphi^c_\ell(\bar{v}_\ell, x_{\ell+1}) := \quad & y_\ell = 0 \wedge \psi_{\ell-1}(\bar{v}_{\ell-1}, 0, z_\ell) \\ & \wedge ((z_\ell = 0 \to x_{\ell+1} = c') \wedge (z_\ell \neq 0 \to R_+ x_\ell \, 1 \, x_{\ell+1})). \end{aligned}$$

The definition for the remaining standard instructions should be clear now.

For the GUESS instruction, i.e., $\pi_c = \text{GUESS}$, we set

$$\varphi^c_\ell(\bar{v}_\ell, x_{\ell+1}) := \exists z(\psi_{\ell-1}(\bar{v}_{\ell-1}, 0, z) \wedge z_i \le z \wedge y_\ell = 0) \wedge R_+ x_\ell \, 1 \, x_{\ell+1}.$$

□

5. The classes of the A-hierarchy

In [10], the classes of the A-hierarchy were introduced. By [11], the following definition is equivalent.

Definition 17. For $t \ge 1$,

$$\mathrm{A}[t] := [p\text{-MC}(\Sigma_t)]^{\mathrm{fpt}}.$$

Note that $\mathrm{W}[1] = \mathrm{A}[1]$ by Definition 13. Again model-checking problems on some restricted classes are already complete for the class $\mathrm{A}[t]$.

Theorem 18 (Flum, Grohe [10]). *For $t \ge 1$,*

(1) $\mathrm{A}[t] = [p\text{-MC}(\mathrm{GRAPH}, \Sigma_t)]^{\mathrm{fpt}}$.

(2) For every relational vocabulary τ of arity ≥ 2,

$$\mathrm{A}[t] \quad = \quad [p\text{-MC}(\Sigma_t[\tau])]^{\mathrm{fpt}}.$$

To capture the classes of the A-hierarchy, we need alternating random access machines. So in addition to the "GUESS" instruction, an *alternating* RAM, or ARAM, also has a "FORALL" instruction. To emphasize the duality, we call the "GUESS" instruction "EXISTS" from now on. Steps of a computation of an ARAM in which EXISTS or FORALL instructions are executed are called *existential steps* or *universal steps*, respectively. They are the *nondeterministic steps*, all other steps are called *deterministic steps*. Acceptance is defined as usual for alternating machines. For an ARAM we generalize the notion of nfpt-program in the obvious way:

Definition 19. A program $\mathbb{P}$ for an ARAM is an afpt-*program*, if there is a computable function f and a polynomial p such that for every input (x, k) with $|x| = n$ the program $\mathbb{P}$ on every run

(1) performs at most $f(k) \cdot p(n)$ steps;

(2) performs at most $f(k)$ nondeterministic steps, i.e., existential or universal ones;

(3) uses at most the first $f(k) \cdot p(n)$ registers;

(4) contains numbers $\leq f(k) \cdot p(n)$ in all registers at every point of the computation.

The sequence of existential and universals steps in a run ρ of an ARAM can be described by a word $q(\rho) \in \{\exists, \forall\}^*$. Let e be a regular expression over the alphabet $\{\exists, \forall\}$. A program $\mathbb{P}$ for an ARAM is *e-alternating*, if for every input and every run ρ of $\mathbb{P}$ on this input, the word $q(\rho)$ belongs to the language of e. For example, a program for an NRAM corresponds to an $\exists^*$-alternating program for an ARAM.

Definition 20. Let $t \geq 1$ and $Q = \exists$ if t is odd and $Q = \forall$ if t is even. A program that is

$$\underbrace{\exists^*\forall^* \ldots Q^*}_{t \text{ blocks}}\text{-alternating}$$

is also called *t-alternating*.

Analogously to Theorem 15, but instead of Σ_1 now using Σ_t, one can prove the following:

Theorem 21. *Let Q be a parameterized problem and $t \geq 1$. Then, Q is in $\mathrm{A}[t]$ if and only if there is a computable function h and a t-alternating* afpt-*program $\mathbb{P}$ for an* ARAM *deciding Q such that for every run of $\mathbb{P}$ all nondeterministic steps are among the last $h(k)$ steps of the computation, where k is the parameter.*

Proof: For $Q \in \mathrm{A}[t]$, we have $Q \leq^{\mathrm{fpt}} p\text{-MC}(\mathrm{GRAPH}, \Sigma_t)$ by Theorem 18. The required afpt-program $\mathbb{P}$ proceeds similarly as the nfpt-program we presented in the proof of Theorem 15: after computing a graph $\mathcal{G}$ and a sentence $\varphi \in \Sigma_t$, the program $\mathbb{P}$ checks whether $\mathcal{G} \models \varphi$, thereby using EXISTS instructions for existential quantifiers and FORALL instructions for universal quantifiers.

Conversely, assume Q is a parameterized problem decided by a t-alternating afpt-program $\mathbb{P} = (\pi_1, \ldots, \pi_m)$ as in the claim of the theorem. It suffices to show that $Q \leq^{\mathrm{fpt}} p\text{-MC}(\Sigma_t)$. We fix an instance (x, k) of Q. The structure $\mathcal{A}$ is defined as in the proof of Theorem 15. We shall define a sentence $\varphi \in \Sigma_t$ such that

$$\mathbb{P} \text{ accepts } (x, k) \iff \mathcal{A} \models \varphi.$$

Again, in order to have a uniform terminology, we say that register 0 is changed to its actual value, if no register is updated. Since in contrast to Theorem 15, the program $\mathbb{P}$ is an alternating program, we have to formalize the acceptance condition in a bottom up fashion. For $\ell < h(k)$, $1 \leq j \leq t$, $1 \leq c \leq m$, we introduce formulas

$$\varphi_{\ell,j,c}(\bar{w}_\ell) \quad \text{and} \quad \psi_\ell(\bar{w}_\ell, y, z),$$

where $\bar{w}_\ell = y_1, z_1, \ldots, y_\ell, z_\ell$ with the meaning in $\mathcal{A}$:

> If on the nondeterministic part (the part beginning with the first EXISTS instruction) of its run on instance (x,k), the program $\mathbb{P}$, so far has performed ℓ steps of the computation, has changed the content of register y_1 to z_1 ,..., the content of register y_ℓ to z_ℓ (in this order) and the actual value of the program counter is c, and if the run is in the jth alternation block, then there is an accepting continuation of this run

and

> if on the nondeterministic part of its run on instance (x,k), the program $\mathbb{P}$, so far has performed ℓ steps of the computation, has changed the content of register y_1 to z_1 ,..., the content of register y_ℓ to z_ℓ, then the content of register y is z,

respectively.

Let c_1 be the instruction number of the first existential step (of $\mathbb{P}$ on input (x,k)). Then, we see that

$$\mathbb{P} \text{ accepts } (x,k) \quad \Longleftrightarrow \quad \mathcal{A} \models \varphi_{0,1,c_1}.$$

We first define the quantifier-free formulas $\psi_\ell(\bar{w}_\ell, y, z)$ by induction on ℓ:

$$\begin{array}{rcl} \psi_0(y,z) & := & \text{Reg}\, yz, \\ \psi_{\ell+1}(\bar{w}_{\ell+1}, y, z) & := & (y = y_{\ell+1} \to z = z_{\ell+1}) \wedge (y \neq y_{\ell+1} \to \psi_\ell(\bar{w}_\ell, y, z)). \end{array}$$

Now we define $\varphi_{\ell,j,c}(\bar{w}_\ell)$ by induction on the length ℓ, starting with maximal ℓ, i.e, $\ell = h(k) - 1$. If $\ell = h(k) - 1$, we set

$$\varphi_{\ell,j,c}(\bar{w}_\ell) \quad := \quad \begin{cases} \psi_\ell(\bar{w}_\ell, 0, 0), & \text{if } \pi_c = \text{STOP (i.e., } \pi_c \text{ is the STOP instruction)}, \\ 0 \neq 0, & \text{otherwise.} \end{cases}$$

Let $\ell < h(k) - 1$. If $\pi_c = \text{STOP}$, then $\varphi_{\ell,j,c}(\bar{w}_\ell) := \psi_\ell(\bar{w}_\ell, 0, 0)$. Suppose $\pi_c = \text{STORE} \uparrow u$ (i.e., $\pi_c =$ "store the number in register 0 in register s, where s is the content of the uth register"). Let d be a constant symbol with $d^{\mathcal{A}} = u$. Then, we let

$$\varphi_{\ell,j,c}(\bar{w}_\ell) \quad := \quad \begin{cases} \exists y \exists z(\psi_\ell(\bar{w}_\ell, d, y) \wedge \psi_\ell(\bar{w}_\ell, 0, z) \wedge \varphi_{\ell+1,j,c+1}(\bar{w}_\ell, y, z)), & \text{if } j \text{ is odd}, \\ \forall y \forall z((\psi_\ell(\bar{w}_\ell, d, y) \wedge \psi_\ell(\bar{w}_\ell, 0, z)) \to \varphi_{\ell+1,j,c+1}(\bar{w}_\ell, y, z)), & \text{if } j \text{ is even}. \end{cases}$$

If $\pi_c = \text{JZERO } c'$ and j is odd, then

$$\varphi_{\ell,j,c}(\bar{w}_\ell) := \exists z \Big(\psi_\ell(\bar{w}_\ell, 0, z) \wedge \big((z = 0 \to \varphi_{\ell+1,j,c'}(\bar{w}_\ell, 0, z)) \wedge (\neg z = 0 \to \varphi_{\ell+1,j,c+1}(\bar{w}_\ell, 0, z)) \big) \Big)$$

and if j is even, then

$$\varphi_{\ell,j,c}(\bar{w}_\ell) := \forall z \Big(\psi_\ell(\bar{w}_\ell, 0, z) \to \big((z = 0 \to \varphi_{\ell+1,j,c'}(\bar{w}_\ell, 0, z)) \wedge (\neg z = 0 \to \varphi_{\ell+1,j,c+1}(\bar{w}_\ell, 0, z)) \big) \Big).$$

The other standard instructions are treated similarly. We give the definition for nondeterministic instructions. So, let $\pi_c = \text{EXISTS}$. Then, we set

$$\varphi_{\ell,j,c}(\bar{v}) \quad := \quad \begin{cases} \exists y \exists z(\psi_\ell(\bar{w}_\ell, 0, y) \wedge z \leq y \wedge \varphi_{\ell+1,j,c+1}(\bar{w}_\ell, 0, z)), & \text{if } j \text{ is odd}, \\ \exists y \exists z(\psi_\ell(\bar{w}_\ell, 0, y) \wedge z \leq y \wedge \varphi_{\ell+1,j+1,c+1}(\bar{w}_\ell, 0, z)), & \text{if } j < t \text{ and } j \text{ is even}, \\ 0 \neq 0, & \text{otherwise}. \end{cases}$$

Similarly, if $\pi_c = \text{FORALL}$, we set

$$\varphi_{\ell,j,c}(\bar{v}) \quad := \quad \begin{cases} \forall y \forall z((\psi_\ell(\bar{w}_\ell, 0, y) \wedge z \leq y) \to \varphi_{\ell+1,j,c+1}(\bar{w}_\ell, 0, z)), & \text{if } j \text{ is even}, \\ \forall y \forall z((\psi_\ell(\bar{w}_\ell, 0, y) \wedge z \leq y) \to \varphi_{\ell+1,j+1,c+1}(\bar{w}_\ell, 0, z)), & \text{if } j \text{ is odd}, \\ 0 \neq 0, & \text{otherwise}. \end{cases}$$

□

The machine characterizations derived so far, yield the following implication (cf. Figure 1):

Corollary 22. *If* FPT = W[P] *then* FPT = A[1] = A[2] = ...

Proof: Assume FPT = W[P]. By induction on $t \geq 1$, we show that FPT = A[t]. This is clear for $t = 1$, since FPT $\subseteq$ A[1] = W[1] $\subseteq$ W[P] = FPT. Now, let Q be a parameterized problem in A[$t+1$] and let $\mathbb{P}$ be an afpt-program for an ARAM according to Theorem 21 deciding Q. Choose f and p according to Definition 19. We show that Q is in W[P] and hence, in FPT.

Fix an instance (x, k) of Q. We stop the program $\mathbb{P}$, on input (x, k), after the existential steps of the first existential block have been performed. Code the contents of the first $f(k) \cdot p(n)$ registers and the value of the program counter by a string y with $|y| \leq O((f(k) \cdot p(n))^2)$. Consider a program $\mathbb{P}'$ that on input (y, k), first decodes y, stores the numbers in the corresponding registers, and sets the program counter accordingly, and then continues the computation of $\mathbb{P}$, where we stopped it. Thus, $\mathbb{P}'$ is a

$$\underbrace{\forall^*\exists^* \ldots Q^*}_{t \text{ blocks}}\text{-alternating}$$

afpt-program for an ARAM such that for some computable function h, all existential and universal steps are among the last $h(k)$ steps of the computation. Therefore, $\mathbb{P}'$ decides a problem whose complement is in A[t] and hence, in FPT by the induction hypothesis A[t] = FPT. Thus, we can replace $\mathbb{P}'$ by an equivalent deterministic afpt-program $\mathbb{P}''$. And by replacing the last t blocks of alternations of $\mathbb{P}$ by $\mathbb{P}''$ appropriately, we get an nfpt-program for an NRAM deciding Q. By Theorem 6, $Q \in$ W[P]. □

5.1. The class AW[P]. We can define the alternating version of p-WSATCIRCUIT as the parameterized problem p-AWSATCIRCUIT:

p-AWSATCIRCUIT	
Input:	A circuit $\mathcal{C}$, $k \in \mathbb{N}$, a partition of the input variables of $\mathcal{C}$ into sets $I_1, \ldots, I_k$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if there is a size k subset J_1 of I_1 such that for every size k subset J_2 of I_2 there exists ... such that the truth assignment setting all variables in $J_1 \cup \ldots \cup J_k$ to TRUE and all other variables to FALSE satisfies $\mathcal{C}$.

AW[P] is the class of all parameterized problems that are fpt-reducible to p-AWSATCIRCUIT, that is,

$$\text{AW[P]} \quad := \quad [p\text{-AWSATCIRCUIT}]^{\text{fpt}}.$$

Generalizing the proof of Theorem 6 to the alternating case, one obtains the following machine characterization of AW[P]:

Theorem 23. *Let Q be a parameterized problem. Then, Q is in* AW[P] *if and only if Q is decided by an* afpt*-program for an* ARAM*.*

6. The classes of the W[func]-hierarchy

Let $t, u \geq 1$. A first-order formula φ is $\Sigma_{t,u}$, if it is Σ_t and all quantifier blocks after the leading existential block have length $\leq u$. For example

$$\exists x_1 \ldots \exists x_k \forall y \exists z_1 \exists z_2 \psi,$$

where ψ is quantifier-free, is in $\Sigma_{3,2}$ (for any $k \geq 1$). In addition, note that $\Sigma_{1,u} = \Sigma_1$ for any u.

Now, the classes of the W-hierarchy can be defined as follows:

Definition 24. For $t \geq 1$,

$$\text{W}[t] \quad := \quad [p\text{-MC}(\Sigma_{t,1})]^{\text{fpt}}.$$

There are several fpt-equivalent variants of p-MC$(\Sigma_{t,1})$ as shown by:

Theorem 25 (Downey, Fellows, Regan [7], Flum, Grohe [10, 11]). *Fix $t \geq 1$.*

(1) $\mathrm{W}[t] = [p\text{-MC}(\mathrm{GRAPH}, \Sigma_{t,1})]^{\mathrm{fpt}}$.

(2) For every relational vocabulary τ of arity ≥ 2, $\mathrm{W}[t] = [p\text{-MC}(\Sigma_{t,1}[\tau])]^{\mathrm{fpt}}$.

(3) For every $u \geq 1$, $\mathrm{W}[t] = [p\text{-MC}(\Sigma_{t,u})]^{\mathrm{fpt}}$.

Comparing the characterizations of $\mathrm{A}[t]$ by model-checking problems for Σ_t in Definition 17 and by programs for alternating machines in Theorem 21, we see an analogy between the blocks of quantifiers on the one hand and the blocks of nondeterministic steps without alternation on the other hand (in fact, this is more than a pure analogy, as it is already known from classical complexity theory). So, the definition of $\mathrm{W}[t]$ suggests to consider $(t,1)$-alternating, where:

Definition 26. Let $t \geq 1$ and $Q = \exists$ if t is odd and $Q = \forall$ if t is even. A program that is

$$\underbrace{\exists^*\forall\exists\ldots Q}_{t \text{ blocks}}\text{-alternating}$$

is also called $(t,1)$-*alternating*.

One would conjecture that the parameterized problems in $\mathrm{W}[t]$ coincide with those decidable on an alternating RAM by $(t,1)$-alternating afpt-programs having the nondeterministic steps in the last part of the computation. In fact, any problem in $\mathrm{W}[t]$ can be decided by such a program; this can be easily seen by slightly modifying the first part of the proof of Theorem 21 (and by using Theorem 25). The converse direction of that proof does not seem to go through: There, we used quantifiers in the corresponding formula also for the deterministic steps of the last part of the computation, so that the length of the quantifier blocks can *not* be bounded in advance. For the deterministic steps the dependence of the quantified variables is functional, hence we could do without additional quantifiers, if we allow vocabularies with function symbols.

Thus, we now consider model-checking problems on structures of *arbitrary* vocabularies, that is, vocabularies that may contain function and constant symbols.

6.1. Arbitrary Vocabularies and Structures. A vocabulary τ is a finite set of relation symbols, function symbols and constant symbols (also called constants). As the relation symbols, every function symbol $f \in \tau$ has an arity, denoted by $\mathrm{arity}(f)$. The arity of τ is the maximum of the arities of all relation and function symbols in τ. Clearly, in a τ-structure $\mathcal{A}$ an r-ary function symbol $f \in \tau$ is interpreted by a function $f^{\mathcal{A}} : A^r \to A$. The size of $\mathcal{A}$ is the number

$$\|\mathcal{A}\| \quad := \quad |\tau| + |A| + \sum_{\substack{R \in \tau \\ \text{a relation symbol}}} \mathrm{arity}(R) \cdot |R^{\mathcal{A}}| + \sum_{\substack{f \in \tau \\ \text{a function symbol}}} |A|^{\mathrm{arity}(f)+1}.$$

The atomic formulas of first-order logic are of the forms $s = s'$ or $Rs_1 \ldots s_r$, where $s, s', s_1, \ldots, s_r$ are *terms* and $R \in \tau$ is an r-ary relation symbol. Terms are either constants of τ, or variables, or of the form $f(s_1, \ldots, s_r)$, where $s_1, \ldots, s_r$ are again terms and $f \in \tau$ is an r-ary function symbol. For $t, u \geq 1$, let Σ_t^{func} and $\Sigma_{t,u}^{\mathrm{func}}$ be defined analogously as Σ_t and $\Sigma_{t,u}$, but now the formulas may also contain function and constant symbols. Clearly, $\Sigma_t \subseteq \Sigma_t^{\mathrm{func}}$ and $\Sigma_{t,u} \subseteq \Sigma_{t,u}^{\mathrm{func}}$. It is not hard to see that for $t \geq 1$, $[p\text{-MC}(\Sigma_t^{\mathrm{func}})]^{\mathrm{fpt}} = [p\text{-MC}(\Sigma_t)]^{\mathrm{fpt}} = \mathrm{A}[t]$. Unfortunately, we do not know the answer to the following:

Question: Is $p\text{-MC}(\Sigma_{t,1}^{\mathrm{func}}) \leq^{\mathrm{fpt}} p\text{-MC}(\Sigma_{t,1})$ for $t \geq 2$?

We define a new hierarchy of parameterized classes:

Definition 27. For $t \geq 1$,

$$\mathrm{W}^{\mathrm{func}}[t] \quad := \quad [p\text{-MC}(\Sigma_{t,1}^{\mathrm{func}})]^{\mathrm{fpt}}.$$

Clearly, $\mathrm{W}^{\mathrm{func}}[t] \subseteq \mathrm{A}[t]$ and consequently, $\mathrm{W}[1] = \mathrm{W}^{\mathrm{func}}[1] = \mathrm{A}[1]$. For $t \geq 2$, we only know that $\mathrm{W}[t] \subseteq \mathrm{W}^{\mathrm{func}}[t] \subseteq \mathrm{A}[t]$, and any strict inclusion will yield PTIME $\neq$ NP. In particular, our question is equivalent to "$\mathrm{W}^{\mathrm{func}}[t] = \mathrm{W}[t]$?"

We aim at a characterization of the class $\mathrm{W}^{\mathrm{func}}[t]$ by means of $(t,1)$-alternating afpt programs. For this purpose, for arbitrary vocabularies, we first derive results analogous to those of Theorem 25:

Theorem 28. *Fix $t, u \geq 1$.*

(1) For a binary function symbol f, we have $\mathrm{W}^{\mathrm{func}}[t] = [p\text{-MC}(\Sigma^{\mathrm{func}}_{t,u}[\{f\}])]^{\mathrm{fpt}}$.

(2) For every vocabulary τ containing a function symbol of arity ≥ 2, $\mathrm{W}^{\mathrm{func}}[t] = [p\text{-MC}(\Sigma^{\mathrm{func}}_{t,u}[\tau])]^{\mathrm{fpt}}$.

(3) $\mathrm{W}^{\mathrm{func}}[t] = [p\text{-MC}(\Sigma^{\mathrm{func}}_{t,u})]^{\mathrm{fpt}}$.

We prove Theorem 28 in several steps. In a first step we show that if we bound the arities of the vocabularies of the input structures, then we can reduce the model-checking problem to a single vocabulary. For this purpose we just code all functions and relation in a single function (recall that by $\Sigma^{\mathrm{func}}_{t,u}[r]$ we denote the class of $\Sigma^{\mathrm{func}}_{t,u}$-formulas in vocabularies of arity $\leq r$):

Lemma 29. *For $r \geq 1$, there is a vocabulary τ such that*

$$p\text{-MC}(\Sigma^{\mathrm{func}}_{t,u}[r]) \quad \leq^{\mathrm{fpt}} \quad p\text{-MC}(\Sigma^{\mathrm{func}}_{t,u}[\tau]).$$

Proof: Let $r \geq 1$. We set $\tau := \{f, g, 0, 1\}$, where f is $(r+1)$-ary, g unary and where 0 and 1 are constants.

Let $\mathcal{A}$ be a structure in an arbitrary r-ary vocabulary σ. Without loss of generality, we can assume that σ contains no constant symbols, that its function symbols $f_1, \ldots, f_m$ and its relation symbols $R_{m+1}, \ldots, R_s$ are all r-ary, and finally that $\{0, 1, \ldots, s\} \subseteq A$.

The τ-structure $\mathcal{B}$ has the same universe as $\mathcal{A}$, i.e., $B := A$. The symbols in τ are interpreted as follows:

– The function $f^{\mathcal{B}} : B^{r+1} \to B$ is defined by

$$f^{\mathcal{B}}(b_1, \ldots, b_r, b) \quad = \quad \begin{cases} f_b^{\mathcal{A}}(b_1, \ldots, b_r), & \text{if } 1 \leq b \leq m\,, \\ 1 & \text{if } m+1 \leq b \leq s \text{ and } R_b^{\mathcal{A}} b_1, \ldots, b_r, \\ 0, & \text{otherwise.} \end{cases}$$

– The function $g^{\mathcal{B}} : B \to B$ is defined by

$$g^{\mathcal{B}}(b) \quad = \quad \begin{cases} b+1, & \text{if } b \in \{0, 1, \ldots, m-1\}, \\ 0, & \text{otherwise.} \end{cases}$$

– $0^{\mathcal{B}} = 0$ and $1^{\mathcal{B}} = 1$.

For $i \in \{1, \ldots, m\}$ define a term

$$l_i \quad := \quad \underbrace{g(\ldots g(g}_{i \text{ times}}(0)) \ldots);$$

clearly the interpretation $l_i^{\mathcal{B}}$ of l_i in $\mathcal{B}$ is the element i.

Now, given any $\varphi \in \Sigma^{\mathrm{func}}_{t,u}[\sigma]$, let $\widetilde{\varphi}$ be the formula obtained from φ by first replacing every term s recursively by

$$\widetilde{s} \quad := \quad \begin{cases} s, & \text{if } s \text{ is a variable,} \\ f(\widetilde{s_1}, \ldots, \widetilde{s_r}, l_i), & \text{if } s = f_i(s_1, \ldots, s_r) \end{cases}$$

and then by replacing atomic subformulas $s_1 = s_2$ by $\widetilde{s_1} = \widetilde{s_2}$ and atomic subformulas $R_j s_1 \ldots s_r$ by $f(\widetilde{s_1}, \ldots, \widetilde{s_r}, l_j) = 1$. It is easy to see that $(\mathcal{A} \models \varphi \iff \mathcal{B} \models \widetilde{\varphi})$; clearly, $\widetilde{\varphi}$ has the same quantifier structure as φ. □

In a second step we show that we can replace an arbitrary vocabulary by a binary one.

Lemma 30. *For any vocabulary τ, there is a binary vocabulary τ' such that*

$$p\text{-MC}(\Sigma^{\text{func}}_{t,u}[\tau]) \quad \leq^{\text{fpt}} \quad p\text{-MC}(\Sigma^{\text{func}}_{t,u}[\tau']).$$

Moreover, we can require that τ' contains no relation symbols and at most one binary function symbol.

Proof: Let $\mathcal{A}$ be a τ-structure. We define a vocabulary τ' of the required form and depending on τ only, and a τ'-structure $\mathcal{A}'$. The universe A' is the union

- of A,
- for each relation symbol $R \in \tau$ of arity r, of the set

  $\{(a_1, \ldots, a_s) \mid 2 \leq s \leq r$, and for some $a_{s+1}, \ldots, a_r \in A$ $(a_1, \ldots, a_s, a_{s+1}, \ldots, a_r) \in R^{\mathcal{A}}\}$,
  i.e., the set of all "partial" tuples that can be extended to some tuple in $R^{\mathcal{A}}$

  (note that the size of this set is bounded by $(r-1) \cdot |R^{\mathcal{A}}|$),
- for each function symbol $f \in \tau$ of arity r, of the set

  $$\bigcup_{2 \leq s \leq r} A^s,$$

  (the size of this set is bounded by $|A|^{r+1}$, thereby we assume that $|A| \geq 2$)
- of $\{\bot\}$, where $\bot$ is an element distinct from all those introduced previously.

Altogether, $|A'| \leq ||\mathcal{A}||^2 + 1$. Now we define the vocabulary τ' and the τ'-structure $\mathcal{A}'$ in parallel:

- τ' contains all constants of τ; they keep their interpretation.
- τ' contains a (new) constant c which is interpreted by an arbitrary element of A
- τ' contains a unary function symbol h and we set

$$h^{\mathcal{A}'}(b) \quad = \quad \begin{cases} c & \text{if } b \in A, \\ \bot, & \text{otherwise.} \end{cases}$$

- For every r-ary relation symbol $R \in \tau$, the vocabulary τ' contains a unary function symbol h_R and we set

$$h^{\mathcal{A}'}(b) \quad = \quad \begin{cases} c & \text{if } b = (a_1, \ldots, a_r) \text{ and } R^{\mathcal{A}} a_1 \ldots a_r \ , \\ \bot, & \text{otherwise.} \end{cases}$$

- For every r-ary function symbol $g \in \tau$, the vocabulary τ' contains a unary function symbol $f_g \in \tau'$ and we define $f_g^{\mathcal{A}'}$ by

$$f_g^{\mathcal{A}'}(b) \quad = \quad \begin{cases} g^{\mathcal{A}}(a_1, \ldots, a_r), & \text{if } b = (a_1, \ldots, a_r), \text{[2]} \\ \bot, & \text{otherwise.} \end{cases}$$

- τ' contains a binary function symbol e; the "tuple *e*xtending function" $e^{\mathcal{A}'}$ is defined by

$$e^{\mathcal{A}'}(b, b') \quad = \quad \begin{cases} (a_1, \ldots, a_s, b'), & \text{if } b = (a_1, \ldots, a_s) \text{ and } (a_1, \ldots, a_s, b') \in A', \\ \bot, & \text{otherwise.} \end{cases}$$

[2] We identify (a_1) with a_1.

Now, for any sentence $\varphi \in \Sigma^{\text{func}}_{t,u}[\tau]$, we construct a sentence $\widetilde{\varphi}$ equivalent to a $\Sigma^{\text{func}}_{t,u}[\tau']$-sentence such that

$$\mathcal{A} \models \varphi \quad \iff \quad \mathcal{A}' \models \widetilde{\varphi}. \tag{2}$$

For this purpose, we first define $\widetilde{s}$ for every term s by induction: If s is a variable or a constant, then $\widetilde{s} := s$. If s is the composed term $g(s_1, \dots, s_r)$, then

$$\widetilde{s} \quad := \quad f_g(e((\dots e(e(\widetilde{s_1}, \widetilde{s_2}), \widetilde{s_3}) \dots), s_r)).$$

For formulas φ we define $\widetilde{\varphi}$ by induction as follows:

$$\begin{aligned} \widetilde{s_1 = s_2} \quad &:= \quad \widetilde{s_1} = \widetilde{s_2}, \\ \widetilde{R s_1 \dots s_r} \quad &:= \quad h_R(e((\dots e(e(\widetilde{s_1}, \widetilde{s_2}), \widetilde{s_3}) \dots), \widetilde{s_r})) = c. \end{aligned}$$

If $\varphi = (\psi_1 \vee \psi_2)$, then $\widetilde{\varphi} := (\widetilde{\psi_1} \vee \widetilde{\psi_2})$, and similarly for formulas $(\psi_1 \wedge \psi_2)$ and $\neg\psi$. If $\varphi = \exists x \psi$ or $\varphi = \forall x \psi$ we define

$$\widetilde{\varphi} := \exists x(h(x) = c \wedge \widetilde{\psi}) \quad \text{or} \quad \widetilde{\varphi} := \forall x(h(x) = c \to \widetilde{\psi}),$$

respectively. Now it is straightforward to verify the equivalence (2); clearly, if $\varphi \in \Sigma^{\text{func}}_{t,u}[\tau]$ then $\widetilde{\varphi}$ is equivalent to a formula in $\Sigma^{\text{func}}_{t,u}[\tau']$. □

The preceding proof yields:

Corollary 31. *For $t, u \geq 1$,*

$$p\text{-MC}(\Sigma^{\text{func}}_{t,u}) \quad \leq^{\text{fpt}} \quad p\text{-MC}(\Sigma^{\text{func}}_{t,u}[2]).$$

In the last step we show:

Lemma 32. *For $t, u \geq 1$ and every vocabulary τ,*

$$p\text{-MC}(\Sigma^{\text{func}}_{t,u}[\tau]) \quad \leq^{\text{fpt}} \quad p\text{-MC}(\Sigma^{\text{func}}_{t,u}[\{f\}])$$

where f is a binary function symbol.

Proof: Let τ be a vocabulary. By Lemma 30, we may assume that τ only contains constants, unary function symbols, and a single binary function symbol. Since for the purpose of our claim we can replace constants by unary function symbols, it suffices to show that

$$p\text{-MC}(\Sigma^{\text{func}}_{t,u}[\{g_1, g_2\}]) \quad \leq^{\text{fpt}} \quad p\text{-MC}(\Sigma^{\text{func}}_{t,u}[\{f\}]),$$

where g_1 is unary and g_2 is binary.

So we have to merge g_1 and g_2 into f. One could add an element $\perp$ to the universe and set $f(x, \perp) = g_1(x)$ and $f(x, y) = g_2(x, y)$ for $x, y \neq \perp$. But, in general, $\perp$ will not be definable by a quantifier-free formula, so that we get problems, for example when relativizing quantifiers to the old universe (at least for $t = 1$). So, we have to define a more involved reduction.

Let $\mathcal{A}$ be a $\{g_1, g_2\}$-structure. We let $\mathcal{A}'$ be the $\{f\}$-structure with universe

$$A' := A \cup \{\perp\} \cup (A \times \{\perp\})$$

and define $f^{\mathcal{A}'}$ by

$$\begin{aligned} f^{\mathcal{A}'}(\perp, a) \quad &:= \quad g_1(a) \text{ for } a \in A; \\ f^{\mathcal{A}'}((a, \perp), (b, \perp)) \quad &:= \quad g_2(a, b) \text{ for } a, b \in A; \\ f^{\mathcal{A}'}(a, \perp) \quad &:= \quad (a, \perp) \text{ for } a \in A; \\ f^{\mathcal{A}'}(a, b) \quad &:= \quad \perp, \text{ otherwise.} \end{aligned}$$

Note that $\bot$ is the only element a of A' with $f^{\mathcal{A}'}(a,a) = a$. Finally, for every $\{g_1, g_2\}$-formula φ we define a $\{f\}$-formula $\widetilde{\varphi}$ such that $(\mathcal{A} \models \varphi \iff \mathcal{A}' \models \widetilde{\varphi})$. For this purpose let u be a new variable. We first define $\widetilde{s}$ for terms by

$$\widetilde{s} \quad := \quad \begin{cases} s, & \text{if } s \text{ is a variable} \\ f(u, \widetilde{s_1}) & \text{if } s = g_1(s_1) \\ f(f(\widetilde{s_1}, u), f(\widetilde{s_2}, u)), & \text{if } s = g_2(s_1, s_2). \end{cases}$$

To obtain $\widetilde{\varphi}$ we replace atomic subformulas $s_1 = s_2$ by $\widetilde{s_1} = \widetilde{s_2}$ and subformulas of the form $\exists x \psi$ or of the form $\forall x \psi$ by $\exists x(\neg f(u,x) = u \wedge \widetilde{\psi})$ and by $\forall x(\neg f(u,x) = u \to \widetilde{\psi})$, respectively. Then, if $\widetilde{\varphi}$ is a $\Sigma^{\text{func}}_{t,u}$-formula, then $\varphi' := \exists u(f(u,u) = u \wedge \widetilde{\psi})$ is equivalent to a $\Sigma^{\text{func}}_{t,u}$-formula and we have $(\mathcal{A} \models \varphi \iff \mathcal{A}' \models \varphi')$. □

Proof of Theorem 28: For $t, u \geq 1$, combining previous results, we obtain the following chain of reductions:

$$\begin{aligned} p\text{-MC}(\Sigma^{\text{func}}_{t,u}) \quad &\leq^{\text{fpt}} \quad p\text{-MC}(\Sigma^{\text{func}}_{t,u}[2]) \quad \text{(by Corollary 31)} \\ &\leq^{\text{fpt}} \quad p\text{-MC}(\Sigma^{\text{func}}_{t,u}[\tau]) \text{ for some vocabulary } \tau \quad \text{(by Lemma 29)} \\ &\leq^{\text{fpt}} \quad p\text{-MC}(\Sigma^{\text{func}}_{t,u}[\{f\}]) \text{ for every binary function symbol } f \quad \text{(by Lemma 32)}. \end{aligned}$$

Since $p\text{-MC}(\Sigma^{\text{func}}_{t,u}) \leq^{\text{fpt}} p\text{-MC}(\Sigma^{\text{func}}_{t,1})$ has been shown as Proposition 8.5 in [10], we obtain all claimed reductions. □

Now we give the machine characterizations of the classes of the W^{func}-hierarchy.

Theorem 33. *Let Q be a parameterized problem and $t \geq 1$. Then, Q is in $\text{W}^{\text{func}}[t]$ if and only if there is a computable function h and a $(t,1)$-alternating* afpt*-program $\mathbb{P}$ for an* ARAM *deciding Q such that for every run of $\mathbb{P}$ all nondeterministic steps are among the last $h(k)$ steps of the computation, where k is the parameter.*

For the proof of this theorem we need the following lemma.

Lemma 34. *For an arbitrary vocabulary τ, let $\varphi_1(\bar{x}), \ldots, \varphi_m(\bar{x})$ and $\psi_1(\bar{x}, \bar{y}), \ldots, \psi_m(\bar{x}, \bar{y})$ be formulas in* FO$[\tau]$*, where $\bar{x} = x_1 \ldots x_r$ and $\bar{y} = y_1 \ldots y_s$ are sequences of variables that have no variable in common. Assume $Q_1, \ldots, Q_s \in \{\forall, \exists\}$. If $\mathcal{A}$ is a τ-structure with $\mathcal{A} \models \forall \bar{x} \neg(\varphi_i \wedge \varphi_j)$ for $i \neq j$ and $\bar{a} \in A^r$, then*

$$\mathcal{A} \models \bigwedge_{i=1}^{m} (\varphi_i \to Q_1 y_1 \ldots Q_s y_s \psi_i)(\bar{a}) \quad \iff \quad \mathcal{A} \models Q_1 y_1 \ldots Q_s y_s \bigwedge_{i=1}^{m} (\varphi_i \to \psi_i)(\bar{a}).$$

We omit the straightforward proof.

Proof of Theorem 33: Let Q be a parameterized problem. Assume first that, for a binary function symbol f,

$$Q \quad \leq^{\text{fpt}} \quad p\text{-MC}(\Sigma^{\text{func}}_{t,1}[\{f\}]).$$

So we have an fpt-algorithm assigning to every instance (x,k) of Q a $\{f\}$-structure $\mathcal{A} = \mathcal{A}_{x,k}$ and a sentence $\varphi = \varphi_{x,k} \in \Sigma_{t,1}[\{f\}]$, say,

$$\varphi \quad = \quad \exists x_{11} \ldots \exists x_{1k_1} \forall y_2 \exists y_3 \ldots P y_t \, \psi,$$

(where $P = \exists$ if t is odd, and $P = \forall$ otherwise) with $|\varphi| \leq g(k)$ for some computable function g, and with a quantifier-free ψ such that

$$Qxk \quad \iff \quad \mathcal{A} \models \varphi.$$

We present a $(t,1)$-alternating afpt-program $\mathbb{P}$ for an ARAM deciding Q. For any input (x,k),

1. it computes the structure $\mathcal{A}$, say, with A an initial segment of the natural numbers, and stores the *array representation* of $f^{\mathcal{A}}$, i.e., for $a_1, a_2 \in A$ a certain register (whose address is easily calculable from a_1, a_2) contains $f^{\mathcal{A}}(a_1, a_2)$;

2. it computes φ;

3. it checks whether $\mathcal{A} \models \varphi$.

To carry out point 3, the program $\mathbb{P}$, using the EXISTS- and FORALL-instructions guesses values of the quantified variables. Then, it checks if the quantifier-free part ψ is satisfied by this assignment. Since we stored the array representation of $\mathcal{A}$, the number of steps needed for point 3 can be bounded by $h(k)$ for some computable h. Hence, all existential and universal steps are among the last $h(k)$ steps of the computation, and the form of the quantifier prefix of φ guarantees that the program $\mathbb{P}$ is $(t,1)$-alternating.

Now let $\mathbb{P} = (\pi_1, \ldots, \pi_m)$ be a $(t,1)$-alternating afpt-program deciding Q such that, for some computable functions f and h and some polynomial p, the program $\mathbb{P}$ on every run on an instance of Q performs at most $f(k) \cdot p(n)$ steps and the nondeterministic ones are among the last $h(k)$ steps.

Fix an instance (x,k). We aim at a structure $\mathcal{A} = \mathcal{A}_{x,k}$ and a $\Sigma^{\text{func}}_{t,1}$-sentence $\varphi = \varphi_{x,k}$ of some vocabulary τ, such that

$$\begin{array}{rcl} Qxk & \iff & \mathbb{P} \text{ accepts } (x,k) \\ & \iff & \mathcal{A} \models \varphi. \end{array}$$

This will give the desired reduction from Q to p-MC$(\Sigma^{\text{func}}_{t,1})$.

Let $\tau := \{\leq, +, -, \text{div}, r, f, 0, d_1, \ldots, d_s\}$ with binary relation symbol $\leq$, with function symbols $+, \cdot, -$ (binary), div, r (unary), and f (4ary), and with the constant symbols $0, d_1, \ldots, d_s$. The structure $\mathcal{A}$ has universe $A := \{0, 1, \ldots, f(k) \cdot p(n)\}$, and moreover

> $\leq^{\mathcal{A}}$ is the natural ordering on A;
> $+^{\mathcal{A}}, -^{\mathcal{A}}$, and div$^{\mathcal{A}}$ are addition, subtraction, and division by two, respectively, restricted to A appropriately;
> for $a \in A$, $r^{\mathcal{A}}(a)$ is the value of the ath register immediately before the first nondeterministic step is carried out;
> $f^{\mathcal{A}}$ corresponds to the "if ... then ... else ..." function, i.e., for $a, b, c, d \in A$,
>
> $$f^{\mathcal{A}}(a,b,c,d) \quad = \quad \begin{cases} c, & \text{if } a = b, \\ d, & \text{if } a \neq b. \end{cases}$$
>
> $0^{\mathcal{A}} = 0$; $d_1^{\mathcal{A}}, \ldots, d_s^{\mathcal{A}}$ are the natural numbers occurring in the program $\mathbb{P}$ as operands.

For $1 \leq j \leq t$, $1 \leq c \leq m$, and terms $i_1, s_1, \ldots, i_\ell, s_\ell$ with $\ell < h(k)$, we introduce a formula

$$\varphi_{j,c,i_1,s_1,\ldots,i_\ell,s_\ell}$$

with the meaning in $\mathcal{A}$ (again we say that register 0 is changed to its actual value, if no register is updated):

> If on the nondeterministic part of its run on instance (x,k), the program $\mathbb{P}$, so far, has performed ℓ steps of the computation, has changed the content of register i_1 to $s_1, \ldots$, the content of register i_ℓ to s_ℓ (in this order) and the actual value of the program counter is c, and if the run is in the jth block without alternation, then there is an accepting continuation of this run

Hence, if c_1 is the instruction number of the first existential step, then

$$\mathbb{P} \text{ accepts } (x,k) \quad \iff \quad \mathcal{A} \models \varphi_{1,c_1}.$$

So it remains to define $\varphi_{j,c,i_1,s_1,\ldots,i_\ell,s_\ell}$. We introduce an abbreviation: For a term i and a sequence of terms $\bar{s} = i_1, s_1, \ldots, i_\ell, s_\ell$, let $a(i,\bar{s})$ be a term denoting the *a*ctual value of the ith register that is, it is $r(i)$, if i differs from all i_j and otherwise, it is s_{j_0} where j_0 is the largest index j with $i = i_j$; we set

$$a(i,\bar{s}) \quad := \quad f(i, i_\ell, s_\ell, f(i, i_{\ell-1}, s_{\ell-1}, f(\ldots f(i, i_1, s_1, r(i)) \ldots))).$$

We define $\varphi_{j,c,\bar{s}}$ with $\bar{s} = i_1, s_1, \ldots, i_\ell, s_\ell$ by induction on the length ℓ, starting with $\ell = h(k) - 1$. For $\ell = h(k) - 1$, we set (recall that, by definition, a run accepts its input, if 0 is the content of the 0th register when stopping)

$$\varphi_{j,c,\bar{s}} \quad := \quad \begin{cases} a(0,\bar{s}) = 0, & \text{if } \pi_c = \text{STOP}, \\ 0 \neq 0, & \text{otherwise.} \end{cases}$$

Let $\ell < h(k) - 1$. If $\pi_c = \text{STOP}$, then again $\varphi_{j,c,\bar{s}} := a(0,\bar{s}) = 0$. Suppose $\pi_c = \text{ADD } u$ (i.e., $\pi_c =$ "Add the numbers stored in the 0th and uth registers and store the result in register 0"). Let d be a constant symbol with $d^{\mathcal{A}} = u$. Then, we let

$$\varphi_{j,c,\bar{s}} \quad := \quad \varphi_{j,c+1,\bar{s},0,a(0,\bar{s})+a(d,\bar{s})}.$$

If $\pi_c = \text{JZERO } c'$, then

$$\varphi_{j,c,\bar{s}} \quad := \quad (a(0,\bar{s}) = 0 \to \varphi_{j,c',\bar{s},0,a(0,\bar{s})}) \wedge (a(0,\bar{s}) \neq 0 \to \varphi_{j,c+1,\bar{s},0,a(0,\bar{s})}). \tag{3}$$

The definition for the other standard instructions is similar. For nondeterministic instructions, say $\pi_c =$ EXISTS, set

$$\varphi_{j,c,\bar{s}} \quad := \quad \begin{cases} \exists x(x \leq a(0,\bar{s}) \wedge \varphi_{j,c+1,\bar{s},0,x}), & \text{if } j = 1, \\ \exists x(x \leq a(0,\bar{s}) \wedge \varphi_{j+1,c+1,\bar{s},0,x}), & \text{if } 1 < j < t \text{ and } j \text{ is even}, \\ 0 \neq 0, & \text{otherwise.} \end{cases}$$

Similarly, if $\pi_c = \text{FORALL}$, we set

$$\varphi_{j,c,\bar{s}} \quad := \quad \begin{cases} \forall x(x \leq a(0,\bar{s}) \to \varphi_{j+1,c+1,\bar{s},0,x}), & \text{if } j < t \text{ and } j \text{ is odd}, \\ 0 \neq 0, & \text{otherwise.} \end{cases}$$

Now, it is easy (using Lemma 34 for subformulas of the form (3)) to verify that φ_{1,c_1} is equivalent to a $\Sigma^{\text{func}}_{t,1}$-sentence. □

6.2. The class AW$[*]$. For $t \geq 1$, the class AW$[t]$ is the alternating version of W$[t]$. It turns out, however, that AW$[t] =$ AW$[1]$ for all $t \geq 1$ (cf. [8]). For that reason, the class AW$[1]$ is usually denoted by AW$[*]$. Using the equalities (cf. [9, 10])

$$\text{AW}[*] = [p\text{-MC(FO)}]^{\text{fpt}} = [p\text{-MC(GRAPH, FO)}]^{\text{fpt}},$$

along the lines of the of the proof of the preceding theorem, one obtains a machine characterization of the class AW$[*]$:

Theorem 35. *Let Q be a parameterized problem. Then, Q is in* AW$[*]$ *if and only if there is a computable function h and an* afpt*-program $\mathbb{P}$ for an* ARAM *deciding Q such that for every run of $\mathbb{P}$ all nondeterministic steps are among the last $h(k)$ steps of the computation.*

7. The classes of the W-hierarchy

In the preceding section, we saw that restricting the programs that characterize the class A$[t]$ in the obvious way in order to obtain programs suitable for W$[t]$, we got a characterization of the class W$^{\text{func}}[t]$, which, for all we know, may be different from W$[t]$. It turns out that for W$[t]$ we have to restrict not only the programs, but also the capabilities of the ARAMs.

An analysis of the proof of the previous theorem reveals that we need function symbols to keep track of the actual values of the registers. Of course, for a deterministic program, at any time of the computation these values only depend on the sequence of instructions carried out (and on the original values of the registers). But, in the presence of nondeterministic steps, the value of *any* register may depend on the guessed numbers. Looking at the first part of the proof of the previous theorem, we see that the guessed numbers represent certain elements of a structure $\mathcal{A}$ and that, in the corresponding program, we did not

need any properties of the guessed numbers but we only wanted to know, if the guessed elements have a given property (say, if the elements $a_1, \ldots, a_r$ are in the relation $R^{\mathcal{A}}$). Therefore, the type of RAM, we are going to introduce, will store the guessed numbers in special registers, called guess registers, and will only have access to the properties of the elements denoted by the guessed numbers. In this way, as in the deterministic case, the values of the *standard* registers only will depend on the sequence of instructions carried out and therefore, function symbols will not be necessary in order to code a run by a $\Sigma_{t,1}$-formula.

We turn to the precise definition of these random access machines that we call WRAMs. A WRAM has

- the *standard registers* $0, 1, \ldots$, their content is denoted by $r_0, r_1, \ldots$, respectively.
- the *guess registers* $0, 1, \ldots$, their contents is denoted by $g_0, g_1, \ldots$, respectively.

Often we denote g_{r_i}, i.e., the contents of the guess register whose index is the content of the ith standard register, by $g(r_i)$.

For the standard registers a WRAM has all the instructions of a standard deterministic random access machine. Moreover, it has four additional instructions:

Instruction	Semantics
EXISTS $\uparrow j$	*existentially* guess a natural number $\leq r_0$; store it in the r_jth guess register
FORALL $\uparrow j$	*universally* guess a natural number $\leq r_0$; store it in the r_jth guess register
JGEQUAL $i\ j\ c$	if $g(r_i) = g(r_j)$, then jump to the instruction with label c
JGZERO $i\ j\ c$	if $r_{\langle g(r_i), g(r_j)\rangle} = 0$, then jump to the instruction with label c.

Here, $\langle\ ,\ \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is any simple coding of ordered pairs of natural numbers by natural numbers such that $\langle i, j\rangle \leq (1 + \max\{i, j\})^2$ and $\langle 0, 0\rangle = 0$. EXISTS instructions and FORALL instructions are the existential and universal instructions, respectively, all the other instructions are said to be deterministic. JGEQUAL instructions and JGZERO instructions are the jump instructions involving guessed numbers.

The following lemma, whose proof is immediate, shows that the contents of the standard registers depend only on the sequence of executed instructions. We already pointed out in the introduction to this section that this property is crucial for the main theorem.

Lemma 36. *Assume that, for a given input, we have two (partial) computations on a* WRAM*. If the same sequence of instructions is carried out in both computations, then the contents of the standard registers will be the same.*

It should be clear what we mean by a $(t, 1)$-alternating afpt-program for a WRAM (cf. Definitions 19 and 26). The promised machine characterization of $\mathrm{W}[t]$ reads as follows:

Theorem 37. *Let Q be a parameterized problem and $t \geq 1$. Then, Q is in $\mathrm{W}[t]$ if and only if there is a computable function h and a $(t, 1)$-alternating* afpt*-program $\mathbb{P}$ for a* WRAM *deciding Q such that for every run of $\mathbb{P}$ all nondeterministic steps are among the last $h(k)$ steps of the computation, where k is the parameter.*

Proof: Let Q be a parameterized problem. First assume that $Q \in \mathrm{W}[t]$. We show that Q is decided by an afpt-program with the claimed properties. The proof is similar to the corresponding part in the proof of Theorem 33. By Theorem 25, we know that

$$Q \quad \leq^{\mathrm{fpt}} \quad p\text{-MC}(\mathrm{GRAPH}, \Sigma_{t,1}).$$

Hence, there is an fpt-algorithm that assigns to every instance (x, k) of Q a graph $\mathcal{G} = \mathcal{G}_{x,k}$ and a sentence $\varphi = \varphi_{x,k} \in \Sigma_{t,1}$, say,

$$\varphi \quad = \quad \exists x_{11} \ldots \exists x_{1k_1} \forall y_2 \exists y_3 \ldots P y_t\, \psi,$$

(where $P = \exists$ if t is odd, and $P = \forall$ otherwise) with $|\varphi| \leq g(k)$ for some computable function g and with a quantifier-free ψ, such that

$$Qxk \quad \Longleftrightarrow \quad \mathcal{G} \models \varphi.$$

The claimed $(t,1)$-alternating afpt-program $\mathbb{P}$ for a WRAM, on input (x,k), proceeds as follows:

1. It computes the graph $\mathcal{G} = (G, E^{\mathcal{G}})$, say, with G an initial segment of the natural numbers, and stores its adjacency matrix in the standard registers as follows:

$$r_{\langle i,j\rangle} = 0 \quad \Longleftrightarrow \quad E^{\mathcal{G}} ij.$$

2. It computes φ.

3. It checks whether $\mathcal{G} \models \varphi$.

To carry out point 3, the program $\mathbb{P}$, using the EXISTS- and FORALL-instructions guesses the values of the quantified variables. Then, it checks the quantifier-free part: JGEQUAL- and JGZERO-instructions are used to check atomic subformulas of the form $x = y$ and of the form Exy, respectively. The number of steps needed for point 3 can be bounded by $h(k)$ for some computable function h. Hence, all existential and universal steps are among the last $h(k)$ steps of the computation.

Now assume that $\mathbb{P}$ is a $(t,1)$-alternating afpt-program for a WRAM deciding Q such that for every instance (x,k), the program $\mathbb{P}$ performs $\leq f(k) \cdot p(n)$ steps with all nondeterministic steps among the last $h(k)$ ones (for some computable functions f and h and polynomial p). We claim that $Q \in \mathrm{W}[t]$. By Theorem 25, it suffices to show that $Q \leq^{\mathrm{fpt}} p\text{-MC}(\Sigma_{t,2})$.

Let $\mathbb{P} = (\pi_1, \ldots, \pi_m)$. We denote instruction numbers by $c, c_1, \ldots$ and finite sequences of instruction numbers by $\bar{c}$. $\ell(\bar{c})$ denotes the last instruction number of the sequence $\bar{c}$, and $[\bar{c})$ the sequence obtained from $\bar{c}$ by omitting its last member. Fix an instance (x,k) of Q. Let

$$C := \bigcup_{0 \leq r < h(k)} \{1, \ldots, m\}^r \quad \text{and} \quad N := \{0, 1, \ldots, f(k) \cdot p(n)\}.$$

We look for a structure $\mathcal{A}$ and a $\Sigma_{t,2}$-sentence φ such that

$$\mathbb{P} \text{ accepts } (x,k) \quad \Longleftrightarrow \quad \mathcal{A} \models \varphi.$$

Let $\bar{c}_1$ be the sequence of instruction numbers of the deterministic part of the run of $\mathbb{P}$ on input (x,k) ending with the instruction number of the next instruction, the first existential instruction to be carried out. As universe A of $\mathcal{A}$ we take $A := C \cup N$. Moreover, in $\mathcal{A}$ there is the binary relation $\leq^{\mathcal{A}}$, the natural ordering on N, and ternary relations $R^{\mathcal{A}}$ and $T^{\mathcal{A}}$ defined by:

$$\begin{aligned}
R^{\mathcal{A}} \bar{c} i j \quad &\Longleftrightarrow \quad \bar{c} \in C,\ i,j \in N, \text{and if } \mathbb{P}, \text{ on input } (x,k), \text{ carries out} \\
&\qquad \text{the sequence of instructions } [\bar{c}_1\, \bar{c}), \text{ then } r_i = j; \\
T^{\mathcal{A}} \bar{c} i j \quad &\Longleftrightarrow \quad \bar{c} \in C,\ i,j, \langle i,j\rangle \in N, \text{and if } \mathbb{P}, \text{ on input } (x,k), \text{ carries out} \\
&\qquad \text{the sequence of instructions } [\bar{c}_1\, \bar{c}), \text{ then } r_{\langle i,j\rangle} = 0.
\end{aligned}$$

Moreover, we have constant symbols, denoted by $0, 1, \ldots, h(k)-1$, for the elements $0, 1, \ldots, h(k)-1$ (so we make use of the analogon of Lemma 16 for $\Sigma_{t,u}$). This finishes the definition of $\mathcal{A}$, which can be constructed within the time allowed by an fpt-reduction.

We turn to the definition of φ. First, we fix $\bar{c} \in C$: Let $i = i(\bar{c})$ be the number of blocks of the sequence of instructions determined by $[\bar{c}_1 \bar{c})$. If $i \leq t$ and if each block determined by $\bar{c}$, besides the first one, contains exactly one nondeterministic step, we introduce a formula

$$\varphi_{\bar{c}}(\bar{x}, x_{h(k)+1}, \ldots, x_{h(k)+i-1}), \tag{4}$$

where $\bar{x} := x_1, \ldots, x_{h(k)}$ with the intuitive meaning in $\mathcal{A}$

if a partial run of $\mathbb{P}$ has $\bar{c}_1\,\bar{c}$ as sequence of instructions numbers and if for $1 \leq j \leq h(k)$, the variable x_j is the value of the jth guess of the first block (and x_j does not occur in $\varphi_{\bar{c}}$, if there was no such guess) and if for $1 \leq j < t$, the variable $x_{h(k)+j}$ is the value of the guess of the jth bounded block, then there is an accepting continuation of this run.

Then, for the empty sequence $\emptyset$ of instruction numbers and $\varphi := \varphi_\emptyset$, we have

$$\mathbb{P} \text{ accepts } (x,k) \quad \Longleftrightarrow \quad \mathcal{A} \models \varphi.$$

For $\bar{c} \in C$ of maximal length, that is, $|\bar{c}| = h(k) - 1$, we set

$$\varphi_{\bar{c}} \quad := \quad \begin{cases} 0 = 0, & \text{if } \pi_{\ell(\bar{c})} = \text{STOP and } R^{\mathcal{A}}\bar{c}\,0\,0, \\ 0 \neq 0, & \text{otherwise.} \end{cases}$$

If $\bar{c} \in C$ and $|\bar{c}| < h(k) - 1$, we assume that $\varphi_{\bar{c}'}$ has already been defined for all $\bar{c}'$ with $|\bar{c}| < |\bar{c}'|$. The definition depends on the type of the instruction of $\mathbb{P}$ with instruction number $\ell(\bar{c})$.

If $\pi_{\ell(\bar{c})} = \text{STOP}$, then again

$$\varphi_{\bar{c}} \quad := \quad \begin{cases} 0 = 0, & \text{if } R^{\mathcal{A}}\bar{c}\,0\,0 \\ 0 \neq 0, & \text{otherwise.} \end{cases}$$

The definition of $\varphi_{\bar{c}}$ is simple for the standard instructions, e.g., if $\pi_{\ell(\bar{c})} = \text{STORE} \uparrow u$ (that is, "$\pi_{\ell(\bar{c})} = r_{r_u} := r_0$"), then

$$\varphi_{\bar{c}} \quad := \quad \varphi_{\bar{c}\,\ell(\bar{c})+1}.$$

We give the definitions for the new instructions: Assume $\pi_{\ell(\bar{c})} = \text{EXISTS} \uparrow v$. Again, let $i = i(\bar{c})$ be the number of blocks of the sequence of instructions determined by $[\bar{c}_1\bar{c})$; if $i = 1$, let $j = j(\bar{c})$ be the number of guesses in this block. We set

$$\varphi_{\bar{c}} \quad := \quad \begin{cases} \exists x_{j+1} \exists y (R\bar{c}\,0\,y \wedge x_{j+1} \leq y \wedge \varphi_{\bar{c}\,\ell(\bar{c})+1}), & \text{if } i = 1, \\ \exists x_{h(k)+i} \exists y (R\bar{c}\,0\,y \wedge x_{h(k)+i} \leq y \wedge \varphi_{\bar{c}\,\ell(\bar{c})+1}), & \text{if } 1 < i < t \text{ and } i \text{ is even}, \\ 0 \neq 0, & \text{otherwise.} \end{cases}$$

The definition is similar for instructions of the type FORALL $\uparrow v$, even easier: define $i = i(\bar{c})$ as before and set

$$\varphi_{\bar{c}} \quad := \quad \begin{cases} \forall x_{h(k)+i} \forall y ((R\bar{c}\,0\,y \wedge x_{h(k)+i} \leq y) \rightarrow \varphi_{\bar{c}\,\ell(\bar{c})+1}), & \text{if } i < t \text{ and } i \text{ is odd}, \\ 0 \neq 0, & \text{otherwise.} \end{cases}$$

Assume $\pi_{\ell(\bar{c})}$ is the instruction JGEQUAL $v\ w\ c$. We need $g(r_v)$ and $g(r_w)$. We determine the actual contents v_0 and w_0 of the vth and the wth standard registers, that is, v_0 and w_0 with $R^{\mathcal{A}}\bar{c}vv_0$ and $R^{\mathcal{A}}\bar{c}ww_0$. Consider the last instructions in $\bar{c}$ of the form FORALL $\uparrow z$ or EXISTS $\uparrow z$ such that at that time $r_z = v_0$, so we can determine the index j of the variable in (4) associated with this guess. Similarly, let $x_{j'}$ be the variable corresponding to the last instruction of the form FORALL $\uparrow z$ or EXISTS $\uparrow z$ such that at that time $r_z = w_0$ (the case that such instructions do not exist is treated in the obvious way). Then, we set

$$\varphi_{\bar{c}} \quad := \quad (x_j = x_{j'} \rightarrow \varphi_{\bar{c}\,c}) \wedge (x_j \neq x_{j'} \rightarrow \varphi_{\bar{c}\,\ell(\bar{c})+1}).$$

Assume $\pi_{\ell(\bar{c})} = \text{JGZERO}\ u\ v\ c$. As in the preceding case, let x_j and $x_{j'}$ denote the actual values of the r_uth and the r_vth guess registers, respectively. Then, we set

$$\varphi_{\bar{c}} \quad := \quad (T\bar{c}x_jx_{j'} \rightarrow \varphi_{\bar{c}\,c}) \wedge (\neg T\bar{c}x_jx_{j'} \rightarrow \varphi_{\bar{c}\,\ell(\bar{c})+1}).$$

As already mentioned above, we set $\varphi := \varphi_\emptyset$. Using Lemma 34, one easily verifies that φ is equivalent to a $\Sigma_{t,2}$-formula. Clearly, the length of φ can be bounded in terms of $h(k)$ and

$$\begin{aligned} Qxy \quad &\Longleftrightarrow \quad \mathbb{P} \text{ accepts } (x,y) \\ &\Longleftrightarrow \quad \mathcal{A} \models \varphi. \end{aligned}$$

This gives the desired reduction from Q to p-MC$(\Sigma_{t,2})$. □

Of course, we could also have used random access machines with restricted access to the guessed numbers, that is, WRAMs, in the preceding sections. First it is easy to see that for every regular expression e over $\{\forall, \exists\}$, every e-alternating afpt-program $\mathbb{P}$ for a WRAM can be simulated by an e-alternating afpt-program $\mathbb{P}'$ for an ARAM. The program $\mathbb{P}'$ first computes the values of the pairing function $\langle\,,\,\rangle$ for inputs $\leq f(k) \cdot p(n)$; then for some constant c, every step of $\mathbb{P}$ is simulated by $\leq c$ steps of $\mathbb{P}'$.

However, to simulate programs for ARAMs by programs for WRAMs, we may need polynomially many steps to figure out what number has been guessed in an EXISTS instruction or an FORALL instruction (recall that ARAMs have direct access to the guessed numbers). We sketch such a procedure for a WRAM; it existentially guesses a number and finally stores it in the standard register 0. For some polynomial p it takes $p(r_0)$ steps (where r_0 is the content of the 0th standard register).

1. $r_1 := 0$.
2. EXISTS $\uparrow 1$ (guess a number $s \leq r_0$ and set $g_{r_1}\,(= g_0) := s$).
3. We use a loop on i from 0 to r_0. In the ith iteration step we make sure that

$$r_{\langle i,i\rangle} = 0 \quad \text{and} \quad \text{for all } 0 \leq j \leq r_0 \text{ with } j \neq i\text{: } r_{\langle j,j\rangle} \neq 0$$

and with the instruction JGZERO $11c$ we will test whether

$$r_{\langle g_{r_1}, g_{r_1}\rangle} = r_{\langle g_0, g_0\rangle} = 0,$$

that is, whether $s = i$. In the positive case, we set $r_0 := i$ and stop.

Using these simulations one can formulate all machine characterizations of the preceding sections in terms of WRAMs; for example, Theorem 21 would read

Theorem 38. *Let $t \geq 1$ and let Q be a parameterized problem. Then, Q is in* A$[t]$ *if and only if there is a computable function h and a t-alternating* afpt*-program $\mathbb{P}$ for a* WRAM *deciding Q such that for every run of $\mathbb{P}$ all nondeterministic steps are among the last $h(k)$ steps of the computation, where k is the parameter.*

7.1. The W*-hierarchy. In [9] the W*-hierarchy was introduced. It is known that W$[1]$ = W*$[1]$ (cf. [9]), W$[2]$ = W*$[2]$ (cf. [5]), W$[t] \subseteq$ W*$[t]$ and that

$$\text{W}^*[t] = [p\text{-MC}(\Sigma^*_{t,1})]^{\text{fpt}} = [p\text{-MC}(\Sigma^*_{t,1})[2]]^{\text{fpt}}$$

(cf. [11], where also the class of formulas $\Sigma^*_{t,1}$ was introduced). Using this last result, one can extend the WRAMs to W*RAMs appropriately in order to get machine characterizations of the classes of the W*-hierarchy. We leave the details to the reader and only remark that two changes are necessary:

– W*RAM are able to existentially and universally guess numbers $\leq k$ (the parameter) and to store them in standard registers;

– for some $\ell \in \mathbb{N}$, instead of the instruction JGZERO ijc, the W*RAMs contain instructions JGm ijc for $m \leq \ell$ with the semantic: if $r_{\langle g(r_i), g(r_j)\rangle} = m$, then jump to the instruction with label c.

8. Conclusions

By giving machine characterizations of many complexity classes of parameterized problems, we feel that we have gained a much clearer understanding of these classes. Now we have a fairly comprehensive picture of the machine side of parameterized complexity theory. The only important classes not yet integrated into this picture are the classes W[SAT] and AW[SAT].

The machine characterization of W[P] is very simple and natural, and provides a precise connection between parameterized complexity theory and limited nondeterminism. When trying to generalize the

machine characterization for the classes of the A-hierarchy to the W-hierarchy, we actually ended up with a new hierarchy, the $\mathrm{W}^{\mathrm{func}}$-hierarchy. We characterize $\mathrm{W}^{\mathrm{func}}[t]$ in terms of the parameterized model-checking problem for $\Sigma_{t,1}$-sentences in vocabularies with function symbols. Of course it would simplify the world of parameterized intractability, if $\mathrm{W}^{\mathrm{func}}[t] = \mathrm{W}[t]$, but at the moment, we even cannot show that $\mathrm{W}^{\mathrm{func}}[t] \subseteq \mathrm{W}[\mathrm{SAT}]$ for $t \geq 2$, while $\mathrm{W}[t] \subseteq \mathrm{W}[\mathrm{SAT}]$.

Our machine characterizations also enable us to investigate some structural issues in parameterized complexity. In particular, we showed that if W[P] = FPT, then the whole A-hierarchy collapses, which can be viewed as a parameterized analogon of the classical result that if PTIME = NP, then the whole *Polynomial Hierarchy* collapses.

References

[1] H.L. Bodlaender, R.G. Downey, M.R. Fellows, M.T. Hallett, and H.T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, 11:49–57, 1995.

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

[3] Y. Chen and J. Flum. Machine characterizations of the classes of the W-hierarchy. In M. Baaz and J. Makowsky, editors, *Proceedings of the 17th International Workshop on Computer Science Logic*, volume 2803 of *Lecture Notes in Computer Science*, pages 114–127. Springer-Verlag, 2003.

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

[5] R.G. Downey and M.R. Fellows. Threshold dominating sets and an improved characterization of $W[2]$. *Theoretical Computer Science*, 209:123–140, 1998.

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

[7] R.G. Downey, M.R. Fellows, and K. Regan. Descriptive complexity and the W-hierarchy. In P. Beame and S. Buss, editors, *Proof Complexity and Feasible Arithmetic*, volume 39 of *AMS-DIMACS Volume Series*, pages 119–134. AMS, 1998.

[8] R.G. Downey, M.R. Fellows, and K. Regan. Parameterized circuit complexity and the W-hierarchy. *Theoretical Computer Science*, 191:97–115, 1998.

[9] R.G. Downey, M.R. Fellows, and U. Taylor. The parameterized complexity of relational database queries and an improved characterization of $W[1]$. In D.S. Bridges, C. Calude, P. Gibbons, S. Reeves, and I.H. Witten, editors, *Combinatorics, Complexity, and Logic – Proceedings of DMTCS '96*, pages 194–213. Springer-Verlag, 1996.

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

[11] J. Flum and M. Grohe. Model-checking problems as a basis for parameterized intractability. To appear in *Proceedings of the 19th IEEE Symposium on Logic in Computer Science*, 2004. Full version available as Technical Report 23/2003, Fakultät für Mathematik und Physik, Albert-Ludwigs-Universität Freiburg, 2003.

[12] G. Gottlob, N. Leone, and M. Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99*, volume 1730 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 1999.

[13] M. Grohe. The parameterized complexity of database queries. In *Proceedings of the 20th ACM Symposium on Principles of Database Systems*, pages 82–92, 2001.

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

[15] C.H. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58:407–427, 1999.

[16] U. Stege. *Resolving Conflicts in Problems from Computational Biology*. PhD thesis, ETH Zuerich, 2000. PhD Thesis No.13364.