Bounded Nondeterminism and Alternation in Parameterized Complexity Theory

Yijia Chen* Jörg Flum† Martin Grohe‡

November 15, 2003

Abstract

We give machine characterisations and logical descriptions of a number of parameterized complexity classes. The focus of our attention is the class W[P], which we characterise 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. We give similar characterisations for AW[P], the "alternating version of W[P]", and various other parameterized complexity classes.

We also give logical characterisations of the classes W[P] and AW[P] in terms of fragments of least fixed-point logic, thereby putting these two classes into a uniform framework that we have developed in earlier work.

Furthermore, we investigate the relation between alternation and space in parameterized complexity theory. In this context, we prove that the COMPACT TURING MACHINE COMPUTATION problem, shown to be hard for the class AW[SAT] in [1], is complete for the class uniform-XNL.

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 analyse problems in various areas of computer science, for example, database theory [16, 21], artificial intelligence [15], and computational biology [3, 22]. 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 [7]). 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], 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 characterisations 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 continue earlier

*Abteilung für Mathematische Logik, Albert-Ludwigs-Universität Freiburg, Eckerstr. 1, 79104 Freiburg, Germany. Email: chen@zermelo.mathematik.uni-freiburg.de

†Abteilung für Mathematische Logik, Albert-Ludwigs-Universität Freiburg, Eckerstr. 1, 79104 Freiburg, Germany. Email: Joerg.Flum@math.uni-freiburg.de.

‡Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK. Email: grohe@inf.ed.ac.uk

[1] Downey and Fellow's monograph [7] 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.

$$\begin{array}{ccccccccccccc}
\mathrm{FPT} \subseteq \mathrm{W}[1] = \mathrm{A}[1] \subseteq \mathrm{W}[2] \subseteq \mathrm{W}[3] \subseteq \cdots & & & & & & \subseteq & \mathrm{W}[\mathrm{SAT}] & \subseteq & \mathrm{W}[\mathrm{P}] \\
& & \cap\mathrel{\mathsf{I}} & & \cap\mathrel{\mathsf{I}} & & & & \cap\mathrel{\mathsf{I}} & & \cap\mathrel{\mathsf{I}} \\
& & \mathrm{A}[2] \subseteq & \mathrm{A}[3] & \subseteq \cdots \subseteq & \mathrm{AW}[*] & \subseteq & \mathrm{AW}[\mathrm{SAT}] & \subseteq & \mathrm{AW}[\mathrm{P}]
\end{array}$$

Figure 1. Parameterized complexity classes.

research [14] in which we try to remedy this situation by giving machine characterisations and logical characterisations of the parameterized complexity classes.

The main focus of this paper is the class W[P] and its "alternating" variant AW[P], two classes we have not considered in our earlier work. W[P] 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. Thus in some sense, W[P] is one of the most natural parameterized complexity classes. W[P] has received some recent attention because of an important result due to Alekhnovich and Razborov [2] showing that resolution is not automatizable unless FPT $=$ W[P]. Our first theorem is a simple machine characterisation of the class W[P]. Intuitively, it 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 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 [7]) 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 characterisation of the class W[1]: A problem is in 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. Here "last steps" means a number of steps that is bounded in terms of the parameter.

The "A classes" and "AW classes" of Figure 1 are defined in terms of an alternating version of the weighted satisfiability problem, which can also be seen as a parameterized version of the satisfiability problem for quantified Boolean formulas. For AW[P], we obtain a similar characterisation as for W[P] in terms of alternating algorithms. Moreover, we get characterisations for the classes A[t], for $t \geq 1$, and the class AW[$*$] that generalise our characterisation of W[1]. A remarkable insight of structural complexity theory is the tight connection between alternation and space [6]. By analogy, it has been suggested to consider the alternating classes AW[$*$], AW[SAT], and AW[P] as "PSPACE-analogues" [7] in the world of parameterized complexity theory. We investigate this idea and obtain a number of results that suggest that the relation between parameter-bounded space and alternation is more complicated than it has been assumed before. Instead of going into further detail here, we refer the reader to the discussion in the introduction of Section 4. The main technical result we prove in that section is that the *compact Turing machine computation* problem, which was only known to be hard for the class AW[SAT] before (cf. [1]), is complete for the class uniform-XNL under parameterized logspace reductions.

Descriptive complexity theory [11, 19] provides a machine independent way of understanding complexity classes. The idea of this area is to characterise the computational complexity of problems in terms of logical definability. Most standard complexity classes have natural descriptive characterisations (for example, [12, 17, 18, 23]). In two earlier papers [13, 14], two of us gave descriptive characterisations of the classes FPT, W[t], and A[t] for $t \geq 1$, and AW[$*$]. Here, we give such characterisations of the classes W[P] and AW[P]. All these characterisations are based on fragments of least fixed point logic, which is the logic that captures polynomial time in classical complexity theory [17, 23]. Our results enable us to place all classes in Figure 1 except W[SAT] and AW[SAT] into a very uniform framework that neither depends on a particular notion of reduction nor on a particular machine model.

2. Parameterized Complexity Theory

We review the notions of parameterized complexity theory most relevant to this paper.

2.1. Fixed-Parameter Tractability. 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*. We always denote the parameter by k and the length of the input string x by n.

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.

2.2. Parameterized Reductions. We shall consider two notions of reductions between parameterized problems in this paper. The first is the standard notion of parameterized (many-one) reduction. We call it FPT-reduction here; Downey and Fellows [7] have used the term strongly uniform parameterized m-reduction. The second, more restrictive notion, called PL-reduction, has been introduced in [14] as a parameterized version of logspace-reduction.

Definition 2. An FPT-*reduction* (PL-*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) There exist a computable function $f : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$ such that R is computable in time $f(k) \cdot n^c$ (computable in space $f(k) + c \cdot \log n$, respectively).

We write $Q \leq^{\text{FPT}} Q'$ ($Q \leq^{\text{PL}} Q'$) if there is an FPT-reduction (a PL-reduction, respectively) from Q to Q'. We let

$$[Q]^{\text{FPT}} = \{Q' \mid Q' \leq^{\text{FPT}} Q\} \quad \text{and} \quad [Q]^{\text{PL}} = \{Q' \mid Q' \leq^{\text{PL}} Q\}.$$

For a class C of parameterized problems, we let

$$[\text{C}]^{\text{FPT}} = \bigcup_{Q \in \text{C}} [Q]^{\text{FPT}} \quad \text{and} \quad [\text{C}]^{\text{PL}} = \bigcup_{Q \in \text{C}} [Q]^{\text{PL}}.$$

Note that $\text{FPT} \subseteq [Q]^{\text{FPT}}$ for every problem $Q \subseteq \Sigma^* \times \mathbb{N}$ with $Q \neq \emptyset, Q \neq \Sigma^* \times \mathbb{N}$.

2.3. Propositional logic. Formulas of propositional logic and circuits are important ingredients in the definitions of various complexity classes of intractable parameterized problems. We recall a few notions and fix our notations: Formulas of propositional logic are built up from *propositional variables* $X_1, X_2, \ldots$ by taking conjunctions, disjunctions, and negations. The negation of a formula α is denoted by $\neg\alpha$. We distinguish between *small conjunctions*, denoted by $\wedge$, which are just conjunctions of two formulas, and *big conjunctions*, denoted by $\bigwedge$, which are conjunctions of arbitrary finite sets of formulas. Analogously, we distinguish between *small disjunctions*, denoted by $\vee$, and *big disjunctions*, denoted by $\bigvee$. A formula is *small* if it only contains small conjunctions and small disjunctions.

SMALL and PROP denote the class of all small and the class of all propositional formulas, respectively. C_1 denote the class of all (big) conjunctions of small formulas, C_2 the class of all conjunctions of disjunctions of small formulas, and C_t for $t \geq 3$ the class of all conjunctions of disjunctions of formulas in C_{t-2}.

The *depth* of a formula is the maximum number of nested conjunctions and disjunctions appearing in this formula. For all $d, t \geq 1$, $C_{t,d}$ denotes the class of all formulas in C_t whose small subformulas have depth at most d (equivalently, we may say that the whole formula has depth at most $d + t$).

We also have to consider *circuits*, defined in the standard way. To be a bit more specific, let us say that our circuits consist of *input gates*, *and gates* and *or gates* of arbitrary finite arity, and *not gates* and they

have one designated output node (that is, they only compute Boolean functions). To simplify our notation, we always assume that we have assigned a propositional variable to each input gate of a circuit. CIRCUIT denotes the class of all circuits. We view PROP as a subclass of CIRCUIT.

2.4. Parameterized complexity classes. The *weight* of an assignment for the variables of a propositional formula or circuit is the number of its variables set to TRUE by the assignment. For any class Θ of propositional formulas or circuits, the *weighted satisfiability problem for* Θ is the problem of deciding whether a formula in Θ has a satisfying assignment of weight k, parameterized by k:

$\mathrm{WSAT}(\Theta)$	
Input:	$\alpha \in \Theta$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Does α have a satisfying assignment of weight k?

Definition 3. (1) For $t \geq 1$, $\mathrm{W}[t]$ is the class of all parameterized problems that are FPT-reducible to $\mathrm{WSAT}(C_{t,d})$ for some $d \geq 1$, that is,

$$\mathrm{W}[t] = [\{\mathrm{WSAT}(C_{t,d}) \mid d \geq 1\}]^{\mathrm{FPT}}.$$

(2) $\mathrm{W}[\mathrm{SAT}]$ is the class of all parameterized problems that are FPT-reducible to $\mathrm{WSAT}(\mathrm{PROP})$, that is,

$$\mathrm{W}[\mathrm{SAT}] = [\mathrm{WSAT}(\mathrm{PROP})]^{\mathrm{FPT}}.$$

(3) $\mathrm{W}[\mathrm{P}]$ is the class of all parameterized problems that are FPT-reducible to $\mathrm{WSAT}(\mathrm{CIRCUIT})$, that is,

$$\mathrm{W}[\mathrm{P}] = [\mathrm{WSAT}(\mathrm{CIRCUIT})]^{\mathrm{FPT}}.$$

For any class Θ of propositional formulas or circuits, the *alternating weighted satisfiability problem for* Θ is the following problem:

$\mathrm{AWSAT}(\Theta)$	
Input:	$\alpha \in \Theta$, $\ell \in \mathbb{N}$, a partition $V_1 \dot\cup \ldots \dot\cup V_\ell$ of the variables of α.
Parameter:	$k, \ell \in \mathbb{N}$.
Problem:	Decide if there is a size k subset U_1 of V_1 such that for every size k subset U_2 of V_2 there exists ... such that the truth assignment setting all variables in $U_1 \cup \ldots \cup U_\ell$ to TRUE and all other variables to FALSE satisfies α.

Definition 4. (1) $\mathrm{AW}[\mathrm{SAT}]$ is the class of all parameterized problems that are FPT-reducible to $\mathrm{AWSAT}(\mathrm{PROP})$, that is,

$$\mathrm{AW}[\mathrm{SAT}] = [\mathrm{AWSAT}(\mathrm{PROP})]^{\mathrm{FPT}}.$$

(2) $\mathrm{AW}[\mathrm{P}]$ is the class of all parameterized problems that are FPT-reducible to $\mathrm{AWSAT}(\mathrm{CIRCUIT})$, that is,

$$\mathrm{AW}[\mathrm{P}] = [\mathrm{AWSAT}(\mathrm{CIRCUIT})]^{\mathrm{FPT}}.$$

Remark 5. Of course there are also classes $\mathrm{AW}[t] = [\{\mathrm{AWSAT}(C_{t,d}) \mid d \geq 1\}]^{\mathrm{FPT}}$ corresponding to the $\mathrm{W}[t]$. It turns out, however, that $\mathrm{AW}[t] = \mathrm{AW}[1]$ for all $t \geq 1$ (cf. [9]). For that reason, the class $\mathrm{AW}[1]$ is usually denoted by $\mathrm{AW}[*]$.

There is another hierarchy, the so called A-hierarchy, whose classes are obtained by fixing the number of alternations (i.e. the ℓ) in the definition of $\mathrm{AW}[*]$. We refer the reader to [13] for the precise definition.

3. Machine descriptions

In this section we derive machine-based characterisations of W[P], AW[P], and of W[1] (= A[1]). The latter one can easily be generalised to the classes of the A-hierarchy. In an informal way, we can describe our results by the following equalities:

$$\begin{aligned}
\text{W[P]} &= \text{parameter-bounded nondeterminism + FPT} \quad \text{(Theorem 8)};\\
\text{W[1]} &= \text{FPT + parameter-bounded nondeterminism} \quad \text{(Theorem 15)};\\
\text{AW[P]} &= \text{parameter-bounded alternating nondeterminism + FPT} \quad \text{(Theorem 17(1))};\\
\text{AW[*]} &= \text{FPT + parameter-bounded alternating nondeterminism} \quad \text{(Theorem 17(2))};\\
\text{A-hierarchy} &= \text{FPT + parameter-bounded alternating nondeterminism}\\
&\quad \text{of bounded alternation depth} \quad \text{(Theorem 17(3))}.
\end{aligned}$$

Our machine model is based on the standard random access machines (RAMs) described in [20]. The arithmetic operations are addition, 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 [20].

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* to be a RAM with an additional instruction "GUESS i j" whose semantics is: Guess a natural number less than or equal to the number stored in register i and store it in register j. Acceptance of an input by a nondeterministic RAM program is defined as usually for nondeterministic machines. Steps of a computation of a nondeterministic RAM 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 12.

Definition 6. A nondeterministic RAM program $\mathbb{P}$ is a *W-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) at most $f(k)$ steps are nondeterministic;

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

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

By standard arguments one gets:

Lemma 7. *Let Q be a parameterized problem. The following are equivalent:*

(1) There is a W-program 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 g and polynomial q).

(3) There is a nondeterministic Turing machine M accepting Q. Moreover, for every $(x, k) \in Q$ there is an accepting run of M of length at most $g(k) \cdot q(n)$ such that the nondeterministic steps are the first $g(k) \cdot \log n$ ones (for some computable g and polynomial q).

Theorem 8. *Let Q be a parameterized problem. Then $Q \in$ W[P] if and only if there is a W-program deciding Q.*

Proof: Assume first that $Q \in$ W[P]. Then by the definition of W[P], $Q \leq^{\text{FPT}}$ WSAT[CIRCUIT]. 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

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

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 W-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". Now, line 2 can be realized by invoking k' times an instruction of the form GUESS i_0 j 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 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 a W-program $\mathbb{P}$. By the previous lemma, there are a computable function f, a polynomial p and a nondeterministic Turing machine M accepting Q such that for every $(x, k) \in Q$ there is a run of M accepting (x, k) of length at most $f(k) \cdot p(n)$ such that the nondeterministic step are the $f(k) \cdot \log n$ first ones. W.l.o.g. 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.

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 [20]) 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 g and 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, 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 $Q \leq^{\text{FPT}}$ WSAT[CIRCUIT], i.e. $Q \in$ W[P]. □

Remark 9. Some of the arguments in the second half of the previous proof have been used by Downey and Fellows [7] 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. [7]). The transition from $\mathcal{C}_{x,k}$ to $\mathcal{D}_{x,k}$ duplicates the proof of [7] showing that W[P] contains SHORT CIRCUIT SATISFIABILITY; there, the method is called the "$k \cdot \log n$ trick".

Note that the reduction of the second part of the proof of the previous theorem is a PL-reduction (parameterized logspace reduction); thus:

Corollary 10. WSAT[CIRCUIT] *is* W[P]*-complete under* PL*-reductions.*

Moreover, by Lemma 7 we obtain:

Corollary 11. *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 polynomial q).*

Remark 12. 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 8 remains true if we allow a W-program to perform $f(k) \cdot \log\ n$ nondeterministic steps (cf. clause (2) in Definition 6).

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 get a slight strengthening of a result of [5]:

Corollary 13. BOUNDED NONDETERMINISTIC MACHINE COMPUTATION (BNTMC) *is* W[P]*-complete under* PL*-reductions.*

Here, (BNTMC) denotes the following problem:

BNTMC	
Input:	A nondeterministic Turing machine M and $n \in \mathbb{N}$ in unary.
Parameter:	$k \in \mathbb{N}$.
Problem:	Does M accept the empty string in at most n steps and using at most k nondeterministic steps?

Proof: BNTMC is in W[P] by Corollary 11: Given a nondeterministic Turing machine M and $n \in \mathbb{N}$ in unary, as input, and $k \in \mathbb{N}$, as parameter, the nondeterministic Turing machine we aim at, guesses the $k \cdot \log\ n$ bits describing the (number and behaviour of the) nondeterministic steps of M and then simulates n steps of M accordingly.

And BNTMC is W[P]-hard: We show that WSAT[CIRCUIT] $\leq^{\text{PL}}$ BNTMC. Given a circuit $\mathcal{C}$ of size n and $k \in \mathbb{N}$ design a nondeterministic Turing machine $M_{\mathcal{C},k}$ whose alphabet, among others, has a letter for every input node of the circuit $\mathcal{C}$ and that first guesses k input nodes to be set to TRUE and then deterministically evaluates the circuit. Hence, for some polynomial q, we have

$$\mathcal{C} \text{ has a satisfying assignment of weight } k \iff ((M_{\mathcal{C},k}, k + q(n)), k) \in \text{BNTMC},$$

which gives the desired reduction. □

We now turn to a machine characterisation of the class W$[1]$. Our proof uses the following result due to Cai, Chen, Downey, and Fellows [4]:

Theorem 14 (Cai et al. [4]). *The following parameterized problem* SHORT TURING MACHINE ACCEPTANCE (STMA) *is* W$[1]$*-complete under* FPT*-reductions:*

STMA	
Input:	A nondeterministic Turing machine M.
Parameter:	$k \in \mathbb{N}$.
Problem:	Does M accept the empty string in at most k steps?

Theorem 15. *Let Q be a parameterized problem. Then $Q \in \mathrm{W}[1]$ if, and only if, there is a computable function h and a W-program $\mathbb{P}$ 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: First assume that $Q \in \mathrm{W}[1]$. Then $Q \leq^{\mathrm{FPT}} \mathrm{STMA}$. Hence, there are a computable function f, a polynomial p, and an algorithm assigning to every instance (x, k) of Q, in time $\leq f(k) \cdot p(n)$, a nondeterministic Turing machine $M = M_{x,k}$ and a natural number $k' = k'(x, k) \leq g(k)$ such that

$$Qxk \quad \Longleftrightarrow \quad M \text{ accepts the empty string in at most } k' \text{ steps.}$$

We can assume that the states and the symbols of the alphabet of M are natural numbers $\leq f(k) \cdot p(n)$. The claimed W-program on (x, k) proceeds as follows:

1. It computes M and k';
2. It guesses a sequence of k' configurations of M;
3. It verifies that the sequence of guessed configurations are an accepting computation of M.

Note that the number of steps needed by line 2 and line 3 is bounded by $h(k)$ for a suitable computable function h; thus, the nondeterministic steps of this program are among the last $h(k)$.

Assume now that the W-program $\mathbb{P}$ 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 $h(k)$. Choose a computable function f and a polynomial p for $\mathbb{P}$ according to the definition of W-program. We show that $Q \leq^{\mathrm{FPT}} \mathrm{STMA}$.

Fix an instance (x, k) for Q. The nondeterministic Turing machine $M = M_{x,k}$ "incorporates in its alphabet and in its transition function the status of the W-program $\mathbb{P}$ immediately before $\mathbb{P}$ carries out its first nondeterministic step and M simulates the nondeterministic part of $\mathbb{P}$." A little bit more in detail:

- The alphabet of M contains as symbols the numbers $0, 1, \ldots, f(k) \cdot p(n)$.
- M has a state c_{cont}: If m_j is the content of register j immediately before the nondeterministic part of $\mathbb{P}$ begins, then M has an instruction

  if M reads j in state c_{cont} then it prints m_j ...
- For every $i \leq f(k) \cdot p(n)$, M has a state $c_{\text{add},i}$ and an instruction

  if M reads $j \leq f(k) \cdot p(n)$ in state $c_{\text{add},i}$ then it prints $i + j$ (in case $i + j \leq f(k) \cdot p(n)$)
  ...
- Let $j \leq h(k)$. If $i_1, \ldots, i_\ell$ are the registers whose content has been changed in the first j steps of the nondeterministic part of $\mathbb{P}$ and $a_1, \ldots, a_\ell$ are their contents after these j steps, then eventually a work-tape of M contains the tuples $(i_1, a_1), \ldots, (i_\ell, a_\ell)$ in any order.

It should be clear that $M = M_{x,k}$ needs $g(h(k))$ steps to simulate the nondeterministic part of $\mathbb{P}$ and that M can be obtained in time $O(f'(k) \cdot p'(n))$ for some computable f' and polynomial p'. Altogether we have an FPT-reduction of Q to STMA. □

3.1. Alternation. To characterise AW[P], AW[$*$], and the classes of the A-hierarchy, we need alternating machines. In addition to the "GUESS i j" instruction, an *alternating RAM* also has a "FORALL i j" instruction. To emphasise the duality, we call the "GUESS i j" instruction "EXISTS i j" from now on. The semantics is defined as usually for alternating machines. Steps of a computation of an alternating RAM in which EXISTS or FORALL instructions are executed are called *existential steps* or *universal steps*, respectively. All other steps are called *deterministic steps*.

Definition 16. An alternating RAM program $\mathbb{P}$ is an *AW-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) at most $f(k)$ steps are existential or universal;

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

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

Analogously to Theorem 8 and Theorem 15, but, instead of STMA, now using the corresponding halting problem for alternating Turing machines (cf. [13]), we can prove the following:

Theorem 17. *Let Q be a parameterized problem.*

(1) Q is in AW[P] *if, and only if, Q is decided by an AW-program.*

(2) Q is in AW[$*$] *if, and only if, there is a computable function h and an AW-program $\mathbb{P}$ deciding Q such that for every run of $\mathbb{P}$ all existential and universal steps are among the last $h(k)$ steps of the computation, where k the parameter.*

(3) For all $t \geq 1$, Q is in A[t] *if, and only if, there is a computable function h and an AW-program $\mathbb{P}$ deciding Q such that for every run of $\mathbb{P}$*

- *all existential and universal steps are among the last $h(k)$ steps of the computation, where k is the parameter,*
- *there are at most $t-1$ alternations between existential and universal states, and the first nondeterministic state is existential.*

4. Parametric Space vs Alternation

In a well-known paper, Chandra, Kozen, and Stockmeyer [6] established a precise connection between alternating time classes and deterministic space classes. In particular, they proved that alternating polynomial time is equivalent to polynomial space.

Wouldn't it be nice if a similar connection held in the world of parameterized complexity theory? Unfortunately, the structure of parameterized complexity classes tends to be more unwieldy than the structure of classical classes, and there is no direct translation between classical and parameterized classes. However, it can be argued that the classes of the W-hierarchy together with W[SAT] and W[P] correspond to NP in classical complexity theory. One reason for this is that all these classes are defined in terms of the NP-complete satisfiability problem. Furthermore, it has turned out that natural parameterizations of NP-complete problems tend to be complete for one of these classes. The characterisations of W[1] and W[P] in terms of nondeterministic machines given in the last section also support this point of view.

If we accept that the W-classes are the parameterized analogue of NP, then we may argue similarly that the AW-classes, that is, weg, denn die A-Hierarchy entspricht in diesem Bild der polynomiellen Hierarchie und nicht alternating PTIME.] AW[$*$], AW[SAT], and AW[P], form a parameterized analogue of alternating polynomial time.

It is now tempting to jump to the conclusion that, since alternating polynomial time is equivalent to polynomial space, the AW-classes can be seen as a parameterized analogue of polynomial space.[2] However, as we want to argue, some care needs to be taken here. One way to support the view that AW corresponds to polynomial space would be to show that the parameterized analogue of a "typical" PSPACE-complete problem is complete for some AW-class. The most generic PSPACE-complete problem, of course, is the space bounded halting problem for Turing machines ("Given a Turing machine and an integer k in unary, does M have an accepting computation that only uses space k."), and this problem has a natural parameterization:

CTMC	
Input:	A deterministic Turing machine M and a string x.
Parameter:	$k \in \mathbb{N}$.
Problem:	Is there an accepting computation of M on input x that visits at most k work tape squares?

[2] Downey and Fellows must have had this in mind when they called the chapter of their book [7] that deals with the AW-classes "Fixed-Parameter Analogs of PSPACE and k-Move Games".

The version of this problem for nondeterministic machines is denoted by CNTMC. Abrahamson, Downey, and Fellows [1] (compare also Theorem 14.4 of [7]) claim that CNTMC is AW[P]-hard under FPT-reductions. Unfortunately, the proof of this result does not seem to be correct.[3] What the proof shows is that CNTMC is hard for AW[SAT] under FPT-reductions. Indeed, as we shall see below, even the deterministic version CTMC is hard for AW[SAT].

We do not know whether this hardness result extends to AW[P], not even for the nondeterministic version, although we tend to believe that this is not the case. We also believe that neither CTMC nor CNTMC are contained in AW[P], but again we have no real evidence to support this believe. Proposition 26 below may be viewed as giving some evidence that at least CTMC and CNTMC are not contained in W[P].

The main result of this section shows that CTMC and CNTMC are complete for a natural parameterized space complexity class derived from the classical deterministic and nondeterministic logarithmic space classes. To define these classes, we need a few more notions from parameterized complexity theory.

A *classical problem* (as opposed to a parameterized problem) is simply a language $R \subseteq \Sigma^*$ over some finite alphabet Σ. For a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ and $k \in \mathbb{N}$, the kth *slice* Q_k is the classical problem $Q_k := \{x \mid (x,k) \in Q\} \subseteq \Sigma^*$. If K is a classical complexity class, XK is the class of parameterized problems all of whose slices are in K. *Uniform*-XK is the class of parameterized problems Q all of whose slices are in K uniformly, that is, there is a computable function assigning to every $k \in \mathbb{N}$ a Turing machine witnessing that $Q_k \in$ K.

It is easy to see that all parameterized complexity classes we have considered so far in this paper are contained in uniform-XP. Here, we are mainly interested in the classes uniform-XL and uniform-XNL derived from the classical classes logarithmic space (denoted by L) and nondeterministic logarithmic space (denoted by NL). It is easy to derive the following alternative characterisations of these classes:

Proposition 18. *Let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. Then:*

(1) Q is in uniform-XL *if, and only if, there is a computable function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that, given a pair $(x,k) \in \Sigma^* \times \mathbb{N}$, decides if $(x,k) \in Q$ in space at most $f(k) \cdot \log(n)$.*

(2) Q is in uniform-XNL *if, and only if, there is a computable function $f : \mathbb{N} \to \mathbb{N}$ and a nondeterministic algorithm that, given a pair $(x,k) \in \Sigma^* \times \mathbb{N}$, decides if $(x,k) \in Q$ in space at most $f(k) \cdot \log(n)$.*

Proof: The backward direction is trivial. For the forward direction, let M be a machine that on input k computes a machine M_k deciding the kth slice Q_k of Q in space at most $c_k \cdot \log n$ (for some constant c_k). M_k is deterministic for (1) and nondeterministic for (2). The desired function f is a computable function such that $f(k) \geq c_k$ and $f(k)$ is an upper bound for the space required by M on input k. □

The following remark is intended for the reader familiar with [14].

Remark 19. A standard diagonalization argument shows that para-NL $\subsetneq$ uniform-XNL. Therefore, by the last proposition there are parameterized problems solvable by some Turing machine and some computable function in space $O(f(k) \cdot \log n)$ but not solvable in space $O(g(k) + \log n)$ for any computable function g. This solves a problem stated in Remark 4 of [14].

Corollary 20. *Both* uniform-XL *and* uniform-XNL *are closed under PL-reductions.*

It is worth noting that FPT $\not\subseteq$ XNL unless PTIME $=$ NL. To see this, take any PTIME-complete (classical) problem R and consider the parameterized problem $Q = \bigcup_{k \in \mathbb{N}} R \times \{k\}$. It is in FPT, but unless PTIME $=$ NL it is not in XNL because the first slice $Q_1 = R$ is not in NL.

Since the closure $[P]^{\text{FPT}}$ of any nontrivial parameterized problem P contains FPT, this also shows that FPT-reductions are not really appropriate when investigating the classes uniform-XL and uniform-XNL. We use PL-reductions instead.[4]

[3] One strong argument showing this is that the reduction described in the proof is actually a PL-reduction. Thus if the reduction would work, then CNTMC would be hard for AW[P] under PL-reductions. By Corollary 25, this would imply that nondeterministic logarithmic space is equivalent to polynomial time.

[4] One may argue that PL-reductions, that is, parameterized *logspace* reductions, are still too powerful when considering the class uniform-XL and that a weaker form of reduction would be more appropriate here, but we do not want to blow up the formal machinery even more, so we simply use PL-reductions.

Theorem 21. *(1)* CTMC *is* uniform-XL-*complete under* PL-*reductions.*

(2) CNTMC *is* uniform-XNL-*complete under* PL-*reductions.*

Before proving part (2) of this theorem (statement (1) can be proved analogously), we fix some notation. A nondeterministic Turing M machine is a tuple $(\Sigma, Q, q_0, q_+, q_-, \delta)$. By default, M has one input tape and one work tape. Σ is the *alphabet* of M, in addition M uses the *blank symbol* $\sqcup$ and the *end marker* $\rhd$; we always assume that $0, 1 \in \Sigma$. Q is the set of *states*, $q_0, q_+, q_- \in Q$ are the *starting state*, the *accepting state*, and the *rejecting state*, respectively. Finally, δ is the *transition relation* consisting of tuples $(q, a, b, q', b', h_1, h_2)$, written in the form $qab \to q'b'h_1h_2$, where $q, q' \in Q$, $a, b, b' \in \Sigma \cup \{\sqcup, \rhd\}$, and $h_1, h_2 \in \{-1, 0, 1\}$ with the obvious meaning.

The *encoding* $\text{enc}(M)$ of M, $\text{enc}(M) \in \{0,1\}^*$, starts with the number $|\Sigma|$ in unary, ended by a 0, followed by $|Q|$ in unary and ended by another 0. And then it indicates the initial, accepting and rejecting states, each by $\log |Q|$ bits. Finally, it has a description of δ, which begins with $|\delta|$ in unary ended by a 0, and then gives the sequence of tuples of δ encoded in a natural way. For example, the ith symbol of $\Sigma \cup \{\sqcup, \rhd\}$ is encoded by the binary representation of i of length $\log(|\Sigma| + 2)$.

The encoding $\text{enc}(x)$ of a string $x \in \Sigma^*$ is the $\{0,1\}$ string consisting of the encoding of the symbols of x and thus has length $|x| \cdot \log(|\Sigma| + 2)$. In a more precise form, CNTMC is the problem:

CNTMC	
Input:	$\text{enc}(M)\text{enc}(x)$,[5] where M is a nondeterministic Turing machine and x a string over its alphabet.
Parameter:	$k \in \mathbb{N}$.
Problem:	Is there an accepting computation of M on input x that visits at most k work tape squares?

By NTIME(f) we denote the set of classical problems decided by a Turing machine in time $c \cdot f$ for some $c \in \mathbb{N}$. The class NSPACE(f) is defined analogously.

By Proposition 18, the following lemma shows that CNTMC $\in$ uniform-XNL.

Lemma 22. CNTMC $\subseteq$ NSPACE($k \cdot \log |\text{enc}(M)\text{enc}(x)|$). *Moreover,* CNTMC($\Sigma$) $\subseteq$ NSPACE($k + \log |\text{enc}(M)\text{enc}(x)|$)*, where* CTMC($\Sigma$) *denotes the problem* CNTMC *restricted to Turing machines with alphabet* Σ*.*

Proof: To simplify the presentation of the argument the desired machine M_0 has 5 work tapes: On input $(\text{enc}(M)\text{enc}(x), k)$, M_0 starts by writing on the first tape the number $s := (|\Sigma| + 2)^k \cdot |Q| \cdot |x| \cdot k$ in binary, i.e., the number of possible configurations of M which only use k squares. Then M_0 simulates M decreasing the value on the first tape by one after the simulation of a step of M. Thereby, it uses the second work tape to record the current state of M, the third and fourth to record the head position of M's input tape and of M's work tape, respectively, and the fifth one to store the actual contents of M's work tape, a string of length $\leq k \cdot \log(|\Sigma| + 2)$. In particular, if M_0 has simulated $s + 1$ steps of M or if the content of its fourth work tape (head position of M's work tape) is already k and M tries to increase it, then M_0 rejects. The space complexity of M_0 is

$$\begin{aligned} & O(\log((|\Sigma| + 2)^k \cdot |Q| \cdot |x| \cdot k)) + \log|Q| + \log|x| + \log k + k \cdot \log(|\Sigma| + 2)) \\ = \;& O(k \cdot \log|\text{enc}(M)| + \log|\text{enc}(x)|) = O(k \cdot \log|\text{enc}(M)\text{enc}(x)|). \end{aligned}$$

For fixed alphabet Σ the first line of the displayed equality shows that the space complexity is $O(k + \log|\text{enc}(M)\text{enc}(x)|)$. □

Proof of Theorem 21: To prove (2), it remains to show that CNTMC is uniform-XNL-hard under PL-reductions. Let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem in uniform-XNL. We shall prove that $Q \leq^{\text{PL}}$ CNTMC.

[5] Due to our encoding we can pinpoint the end position of $\text{enc}(M)$ in $\text{enc}(M)\text{enc}(x)$ easily.

By Proposition 18 (2) there are a computable function f and a nondeterministic Turing machine M that decides Q in space $f(k) \cdot \log n$. Note that the alphabet of this machine is $\Sigma' = \Sigma \cup \{(,), ,\}$ (the last comma is an element of the set, that is, a symbol of Σ'). Then

$$(x,k) \in Q \iff (\text{enc}(M)\text{enc}((x,k)), f(k) \cdot \log |x|) \in \text{CNTMC},$$

and this equivalence suggests a reduction from Q to CNTMC. Unfortunately, this reduction is not a parameterized reduction, because there is no computable function g with $f(k) \cdot \log |x| \leq g(k)$ as required in the definition of PL-reduction. To overcome this problem, given x we use the technique of tape compression where each tape square holds a symbol (of a new alphabet) representing the contents of a string over Σ' of length $\log |x|$ and where the new machine by its instructions keeps track of which symbol of Σ', among those represented by a new symbol, is actually scanned by M. The following data of tape compression are relevant to us.

Let $|x| = n$. The new machine M_n has alphabet $\Sigma_n := (\Sigma' \cup \{\sqcup, \rhd\})^{\log n}$ and set of states $Q_n := Q \times \{0, \ldots, \log n - 1\} \times \{0, \ldots, \log n - 1\}$. Thus, $|\Sigma_n| := (|\Sigma'| + 2)^{\log n} = n^{\log (|\Sigma'|+2)}$ and $|Q_n| = |Q| \cdot (\log\ n)^2$. Hence, $|M_n| \leq |M| \cdot n^c$ for some constant c not depending on n. Denote by com the compression map, com : $(\Sigma')^* \to \Sigma_n^*$. Then

$$\begin{aligned}
(x,k) \in Q &\iff M \text{ accepts } (x,k) \text{ using space} \leq f(k) \cdot \log |x| \\
&\iff M_{|x|} \text{ accepts com}((x,k)) \text{ using space} \leq f(k) \\
&\iff (\text{enc}(M_{|x|})\text{enc}(\text{com}((x,k))), f(k)) \in \text{CNTMC}.
\end{aligned}$$

Altogether this gives a PL-reduction of Q to CNTMC. □

Proposition 23. AWSAT(PROP) $\in$ uniform-XL *and thus* AW[SAT] $\subseteq$ $\big[\text{uniform-XL}\big]^{\text{FPT}}$.

Proof: It is easy to see (and well-known) that propositional formulas can be evaluated in logarithmic space. More precisely, there is an algorithm $\mathbb{A}$ that, given a formula $\alpha \in$ PROP and a truth value assignment T for the variables of this formula, decides in space $O(\log n)$ whether T satisfies α.

Recall that an instance of AWSAT(PROP) consists of a formula $\alpha \in$ PROP, positive integers k, ℓ, and a partition $I_1, \ldots, I_\ell$ of the set of variables of α. Let v be the number of variables of α, m the size of α. Storing an assignment to the variables of α in which for every i exactly k of the variables in I_i are set to TRUE requires space $k \cdot \ell \cdot \log v$. Using the algorithm $\mathbb{A}$ as a subroutine, it is easy to design an algorithm solving AWSAT(PROP) in space $O(k \cdot \ell \cdot \log v + \log m)$. □

Corollary 24. CTMC *is hard for* AW[SAT] *under* FPT*-reductions.*

Corollary 25. *Assume* NL $\neq$ PTIME. *Then* CNTMC *is not* FPT*-hard under* PL*-reductions.*

Proof: Suppose for contradiction that CNTMC is FPT-hard under PL-reductions. Then by Theorem 21 (2), FPT $\subseteq$ uniform-XNL. But we have already noted (on page 10) that FPT $\not\subseteq$ XNL unless PTIME = NL. □

The next result may be seen as giving some evidence that at least CTMC and CNTMC are not contained in W[P].

Proposition 26. *If* CTMC $\in$ W[P] *then there is an* r *such that* L $\subseteq$ NTIME(n^r).
The analogous result holds for CNTMC *with* NL *instead of* L.

Proof: Suppose that CTMC $\in$ W[P]. Then by Theorem 21, uniform-XL $\subseteq$ W[P]. We shall prove that L $\subseteq$ NTIME(n^r) for some $r \geq 1$.

Since for any alphabet Σ and $R \subseteq \Sigma^*$ with $R \in$ L, we have $R^{\{0,1\}} := \{\text{enc}(x) \mid x \in R\} \in$ L, it suffices to consider Turing machines with alphabet $\{0, 1\}$. Let

$$(M_0, c_0), (M_1, c_1), \ldots$$

be a computable enumeration of the pairs (M, c), where M is a deterministic Turing machine with alphabet $\{0,1\}$ and $c \in \mathbb{N}$. We set

$$L(k) \quad := \quad \{x \in \{0,1\}^* \mid M_k \text{ accepts } x \text{ using space} \leq c_k \cdot \log |x|\}$$

and

$$Q \quad := \quad \bigcup_{k \in \mathbb{N}} L(k) \times \{k\}.$$

Clearly, $Q \in$ uniform-XL and thus by our overall assumption, $Q \in$ W[P]. Then by Corollary 11, $Q \in$ NTIME$(f(k) \cdot n^r)$ for some computable function f and constant r. Thus for every k, $Q_k \in$ NTIME(n^r). □

Remark 27. We state a stronger version of Proposition 26 showing that its converse also holds. Recall that para-NP (cf. [14]) is the nondeterministic analogue of FPT, that is, a parameterized problem Q is in para-NP if and only if there is a nondeterministic algorithm accepting Q in time $g(k) \cdot p(n)$ for some computable g and some polynomial p. Clearly, W[P] $\subseteq$ para-NP (e.g., apply Corollary 11).

Then the following three statements are equivalent:

(1) CNTMC $\in$ para-NP.

(2) uniform-XNL $\subseteq$ para-NP.

(3) There is an $r \in \mathbb{N}$ such that NL $\subseteq$ NTIME(n^r) and this inclusion holds in an effective way, i.e., there is an algorithm that, given a Turing machine M and $c \in \mathbb{N}$, yields a Turing machine M' and $c' \in \mathbb{N}$ such that

> if M is $c \cdot \log n$ space-bounded then M' is $c' \cdot n^r$ time-bounded and M and M' accept the same language.

To conclude this section, let us return to our original question of how alternation in parameterized complexity relates to space. In particular, we were interested in whether the problem CTMC, which we have seen to be hard for AW[SAT], is also contained in AW[SAT] or at least in AW[P]. By Theorem 21, the latter would imply that uniform-XL $\subseteq$ AW[P]. We do not believe that this is the case. The intuitive reason for this is that alternation in AW[P] is *parameter-bounded*, but that to simulate a space-bounded computation of length m by an alternating machine one needs about log m quantifier alternations.

We can turn this argument around and show that if we have enough alternations, then we indeed get a problem that is hard for uniform-XNL.

LAWSAT	
Input:	$\alpha \in$ PROP, $\ell, k \in \mathbb{N}$ with $\ell \leq k \cdot \log \|\alpha\|$, a partition $V_1 \dot\cup \ldots \dot\cup V_\ell$ of the variables in α.
Parameter:	$k \in \mathbb{N}$.
Problem:	Decide if there is a size k subset U_1 of V_1 such that for every size k subset U_2 of V_2 there exists ... such that the truth assignment setting all nodes in $U_1 \cup \ldots \cup U_\ell$ to TRUE and all other input nodes to FALSE satisfies α.

Theorem 28. *(1)* LAWSAT *is hard for* uniform-XNL *under* PL*-reductions.*

(2) LAWSAT *is contained in* uniform-XDSPACE$(\log^2 n)$.

Proof: Let an instance (i.e., an input and a parameter) of LAWSAT be given as in its definition. We use the following terminology: If the instance is in LAWSAT we say that the quantified propositional formula

$$\exists V_1 \forall V_2 \ldots Q V_\ell \, \alpha$$

(with $Q = \forall$ for even ℓ, and $Q = \exists$ for odd ℓ) holds in the k-interpretation.

To prove (1), by Theorem 21 it suffices to show that there is a PL-reduction of CNTMC to LAWSAT. For this purpose let $((M, x), k)$ (more precisely $((\text{enc}(M)\text{enc}(x)), k)$ be an instance of CNTMC and denote by n the length of $\text{enc}(M)\text{enc}(x)$. Let c be the number of k space-bounded configurations of the nondeterministic Turing machine M (on inputs of length n). Then $c \leq n^{d \cdot k}$ for some constant d.

To describe such a configuration we use propositional variables $X(i, a), \text{HI}(i), \text{HW}(i)$, and $S(q)$ with $1 \leq i \leq n$ and where a and q range over the alphabet and the set of states of M, respectively. Their intended meaning is:

$$\begin{aligned} X(i,a): &\quad \text{the } i\text{th cell of the work tape contains the letter } a;\\ \text{HI}(i): &\quad \text{the head of the input tape scans the } i\text{th cell;}\\ \text{HW}(i): &\quad \text{the head of the work tape scans the } i\text{th cell;}\\ S(q): &\quad M \text{ is in state } q. \end{aligned}$$

Let C be the set of these propositional variables. In the following, if we write C' or C_j (with $j \in \mathbb{N}$) we mean the set of the corresponding primed or indexed variables.

One easily writes down a propositional formula $\beta(C)$ such that their satisfying assignments of weight $k + 3$ correspond to k space-bounded configurations of M in a natural way.

By induction on m, we define a quantified propositional formula $\Gamma_m(C, C')$ expressing that there is a computation of length $\leq 2^m$ leading from the configuration C to the configuration C' (and where all intermediate configurations are k space-bounded):

$$\begin{aligned} \Gamma_0(C, C') &\quad \text{is a propositional formula of the form } \beta(C) \wedge \beta(C') \wedge \gamma(C, C') \text{ and}\\ &\quad \text{of length } O(k \cdot n);\\ \Gamma_{m+1}(C, C') &= \text{“}\exists C_1(\Gamma_m(C, C_1) \wedge \Gamma_m(C_1, C'))''\\ &= \exists C_1(\forall C_2 \forall C_3((C_2 = C \wedge C_3 = C_1) \vee (C_2 = C_1 \wedge C_3 = C')\\ &\qquad \rightarrow \Gamma_m(C_2, C_3))). \end{aligned}$$

The last line guarantees that $|\Gamma_m(C, C')| \in O(m \cdot k \cdot n)$. We set

$$\Gamma := \exists C \exists C'(\beta_{\text{init}}(C) \wedge \beta_{\text{accept}}(C') \wedge \Gamma_{d \cdot k \cdot \log n}(C, C')),$$

where $\beta_{\text{init}}(C)$ and $\beta_{\text{accept}}(C)$ express that C is the initial or an accepting configuration of M, respectively.

Moving all quantifiers to the front of the formula and adding dummy variables where necessary, one obtains an equivalent formula $\Gamma' = \Gamma'_{((\text{enc}(M)\text{enc}(x)),k)}$ of the form

$$\Gamma' = \exists C_1 \forall C_2 \exists C_3 \ldots Q C_\ell \, \gamma$$

where $\ell \in O(k \cdot \log n)$ and such that

$$((\text{enc}(M)\text{enc}(x)), k) \in \text{CNTMC} \quad \Longleftrightarrow \quad \Gamma' \text{ holds in the } k + 3 \text{ interpretation.}$$

By the introductory remark this gives the desired reduction of CNTMC to LAWSAT (if necessary, we can ensure that $\ell \leq (k + 3) \cdot |\gamma|$, by adding, for some constant e, up to n^e many trivial conjunctions to $\Gamma_0(C, C')$).

It is easy to check that this reduction indeed is a PL-reduction.

For (2) note that LAWSAT considered as a classical problem is in DSPACE($k^2 \cdot \log^2 n$) and hence, the parameterized problem LAWSAT is in uniform-XDSPACE($\log^2 n$). □

5. Descriptive Complexity

The main results of this section are logical descriptions of the classes W[P] and AW[P]. We need a few preliminaries from logic. For more details on the notions which are briefly described in the next subsection we refer the reader to [11].

5.1. Structures and Logic. A *vocabulary* is a finite set of relation, function, and constant symbols. Each relation and function symbol has an *arity*. τ always denotes a vocabulary. A *structure* $\mathcal{A}$ of vocabulary τ, or τ-structure, consists of a set A called the universe, and an interpretation $T^{\mathcal{A}}$ of each symbol $T \in \tau$: Relation symbols and function symbols are interpreted by relations and functions on A of the appropriate arity, and constant symbols are interpreted by elements of A. *We only consider structures whose universe is finite.*

Let $\tau' \subseteq \tau$ be vocabularies. The *τ'-reduct* of a τ-structure $\mathcal{A}$ is the τ'-structure with the same universe as $\mathcal{A}$ that coincides with $\mathcal{A}$ on all symbols in τ'. A *τ-expansion* of a τ'-structure $\mathcal{A}'$ is a τ-structure $\mathcal{A}$ such that $\mathcal{A}'$ is the τ'-reduct of $\mathcal{A}$. If $\mathcal{A}$ is a τ-structure and $B \subseteq A^k$, we often write $(\mathcal{A}, B)$ to denote the $\tau \cup \{R\}$-expansion of $\mathcal{A}$ in which R is interpreted by B. Here R is a k-ary relation symbol not contained in τ that is understood from the context. Similarly, we write $(\mathcal{A}, a_1, \ldots, a_k)$ or just $(\mathcal{A}, \bar{a})$ to denote the expansion of $\mathcal{A}$ by a tuple of constants.

Example 29. (1) *Directed graphs* may be viewed as structures $\mathcal{G} = (G, E^G)$ whose vocabulary consists of one binary relation symbol E.

(2) Let $\tau_{\text{circ}} = \{E, I, G_\wedge, G_\vee, G_\neg, \text{out}\}$, where E is a binary relation symbol, I, $G_\wedge$, $G_\vee$, $G_\neg$ are unary relation symbols, and out is a constant symbol. Boolean circuits may be viewed as τ_{circ}-structures

$$\mathcal{C} = (C, E^{\mathcal{C}}, I^{\mathcal{C}}, G_\wedge^{\mathcal{C}}, G_\vee^{\mathcal{C}}, G_\neg^{\mathcal{C}}, \text{out}^{\mathcal{C}}),$$

where $(C, E^{\mathcal{C}})$ is the directed acyclic graph underlying the circuit, $I^{\mathcal{C}}$ is the set of all input nodes, $G_\wedge^{\mathcal{C}}$, $G_\vee^{\mathcal{C}}$, $G_\neg^{\mathcal{C}}$ are the sets of and-gates, or-gates, and negation-gates, respectively, and $\text{out}^{\mathcal{C}}$ is the output node.

Let T be another unary relation symbol. If $\mathcal{C}$ is a circuit and $T^{\mathcal{C}} \subseteq I^{\mathcal{C}}$, then we may interpret the $\tau_{\text{circ}} \cup \{T\}$-expansion $(\mathcal{C}, T^{\mathcal{C}})$ of $\mathcal{C}$ as a representation of the circuit $\mathcal{C}$ together with the truth value assignment that sets precisely the input nodes in $T^{\mathcal{C}}$ to TRUE.

For every vocabulary τ we let $\tau^{\text{ord}} = \tau \cup \{\leq, S, \min, \max\}$, where $\leq$ is a binary relation symbol, S a unary function symbol, and min and max are constant symbols. An *ordered τ-structure* is a τ^{ord}-structure $\mathcal{A}$ such that $\leq^{\mathcal{A}}$ is a linear order of A, $\min^{\mathcal{A}}$ and $\max^{\mathcal{A}}$ are the minimum and maximum element of $\leq^{\mathcal{A}}$, and $S^{\mathcal{A}}$ is the successor function associated with $\leq^{\mathcal{A}}$, where we let $S^{\mathcal{A}}(\max^{\mathcal{A}}) = \max^{\mathcal{A}}$. By $\text{ORD}[\tau]$ we denote the class of all ordered τ-structures.

We distinguish between the size of the universe A of a τ-structure $\mathcal{A}$, which we denote by $|A|$, and the *size* of $\mathcal{A}$, which is defined to be

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

where $\|\tau\|$ denotes the size (of a natural encoding) of τ.

The formulas of *first-order logic* of vocabulary τ are built up from *atomic formulas* using the Boolean connectives $\neg$, $\wedge$, and $\vee$, and existential and universal quantification (over the elements of the universe of a structure). Remember that an *atomic formula* is a formula of the form $t = u$ or $Rt_1 \ldots t_r$, where $R \in \tau$ is an r-ary relation symbol and $t, u, t_1, \ldots, t_r$ are *terms* formed from *variables* and constant symbols using function symbols. FO denotes the class of formulas of first-order logic. A *sentence* is a formula φ in which every variable is bound by a quantifier. If $\varphi(x_1, \ldots, x_k)$ is a formula with free variables among $x_1, \ldots, x_k$, $\mathcal{A}$ a structure of the same vocabulary, and $(a_1, \ldots, a_k) \in A^k$ a k-tuple of elements of $\mathcal{A}$, we write $\mathcal{A} \models \varphi(a_1, \ldots, a_k)$ to denote that $\mathcal{A}$ satisfies φ if the variables $x_1, \ldots, x_k$ are interpreted by $a_1, \ldots, a_k$, respectively.

The results in this section are mainly concerned with an extension of first-order logic called *least fixed-point logic*, or FO(LFP) for short. To introduce least fixed-point logic, we first observe that any formula $\varphi(x_1, \ldots, x_k)$ of vocabulary $\tau \cup \{X\}$, where X is a k-ary relation symbol not contained in τ, for every τ-structure $\mathcal{A}$ defines an operator $F_\varphi^{\mathcal{A}} : \text{Pow}(A^k) \to \text{Pow}(A^k)$ given by

$$F_\varphi^{\mathcal{A}}(B) = \big\{(a_1, \ldots, a_k) \in A^k \;\big|\; (\mathcal{A}, B) \models \varphi(a_1, \ldots, a_k)\big\}$$

for every $B \subseteq A^k$.

If X only occurs *positively* in φ, that is, only in the scope of an even number of negation symbols, then for every structure $\mathcal{A}$ the operator $F_\varphi^{\mathcal{A}}$ is *monotone* and therefore has a *least fixed point*, which we denote by $\mathrm{lfp}(F_\varphi^{\mathcal{A}})$.

Now the formulas of FO(LFP) are formed by the same rules as the formulas of FO and the following additional formula formation rule: If X is a k-ary relation symbol not contained in the vocabulary τ, $\bar{x} = (x_1, \ldots, x_k)$ and $\bar{y} = (y_1, \ldots, y_k)$ are k-tuples of variables, and φ is a formula of vocabulary $\tau \cup \{X\}$ such that X only occurs positively in φ, then

$$\psi(\bar{y}) = [\mathrm{LFP}_{\bar{x},X}\varphi](\bar{y})$$

is a new formula of vocabulary τ. To define the semantics of $\psi(\bar{y})$, for any τ-structure $\mathcal{A}$ and k-tuple $\bar{b} \in A^k$ of elements of $\mathcal{A}$, we let $\mathcal{A} \models \psi(\bar{b})$ iff $\bar{b} \in \mathrm{lfp}(F_\varphi^{\mathcal{A}})$.

Although we suppress this in our notation to simplify matters, let us point out that the subformula φ of $[\mathrm{LFP}_{\bar{x},X}\varphi](\bar{y})$ may have additional free variables besides those appearing in $\bar{x}$; these are simply treated as free variables of the whole formula. Moreover, the tuple $\bar{y}$ may contain arbitrary terms and not just variables. For a more thorough introduction to least fixed point logic we refer the reader to [11].

$\|\varphi\|$ always denotes the size (of a natural encoding) of a first-order or least fixed-point formula φ.

Example 30. In this example, we show how to define the *monotone circuit value problem* in FO(LFP). A *monotone circuit* is a circuit without negation gates. Recalling Example 29, we may view a monotone circuit as a structure of vocabulary $\tau_{\text{mon-circ}} = \tau_{\text{circ}} \setminus \{G_\neg\}$. As in this example, we view $\tau_{\text{mon-circ}} \cup \{T\}$-expansions $(\mathcal{C}, T^{\mathcal{C}})$ of circuits $\mathcal{C}$ by sets $T^{\mathcal{C}} \subseteq I^{\mathcal{C}}$ as circuits together with truth value assignments.

Let $\mathcal{C}$ be a monotone circuit and $T^{\mathcal{C}} \subseteq I^{\mathcal{C}}$. We observe that the set of all nodes of $\mathcal{C}$ that evaluate to TRUE under the truth value assignment that sets precisely the input nodes in $T^{\mathcal{C}}$ to TRUE is the least set TRUE of nodes such that:

- TRUE contains $T^{\mathcal{C}}$.
- If all children of an and-gate $a \in G_\wedge^{\mathcal{C}}$ are in TRUE then a is in TRUE. (The *children* of a are all $b \in C$ such that $(a, b) \in E^{\mathcal{C}}$.)
- If at least one child of an or-gate $a \in G_\vee^{\mathcal{C}}$ is in TRUE then a is in TRUE.

A moment's thought reveals that therefore TRUE is the least fixed-point of the operator $F_\varphi^{(\mathcal{C},T^{\mathcal{C}})}$ associated with the formula

$$\varphi(x) = Tx \vee \big(G_\wedge x \wedge \forall y(Exy \to Xy)\big) \vee \big(G_\vee x \wedge \exists y(Exy \wedge Xy)\big).$$

Thus TRUE is the set of all nodes b such that

$$(\mathcal{C}, T^{\mathcal{C}}) \models [\mathrm{LFP}_{x,X}\varphi](b).$$

This means that the circuit $\mathcal{C}$ evaluates to TRUE under the assignment represented by $T^{\mathcal{C}}$ if, and only if,

$$(\mathcal{C}, T^{\mathcal{C}}) \models [\mathrm{LFP}_{x,X}\varphi](\mathrm{out}). \tag{2}$$

This gives us the desired definition of the monotone circuit value problem in FO(LFP). Let us point out that the circuit value problem for arbitrary circuits is also definable in FO(LFP). It requires a more complicated formula, though.

5.2. Logical Descriptions of W[P] and AW[P]. In [13] and [14] descriptive characterisations of various parameterized complexity classes were derived, including the classes of the W- and of the A-hierarchy. Here we give similar characterisations of the classes W[P] and AW[P]. The importance of such descriptive characterisations lies in the fact that they neither depend on a particular machine model nor on a particular complete problem, the latter being particularly important for such parameterized complexity classes that are defined in terms of complete problems. In addition, the logical descriptions allow it to translate open

complexity-theoretic problems such as W[P] = W[1] into purely logical problems on the expressive power of logics (compare Corollary 24 in [14]).

In descriptive complexity theory, algorithmic problems are considered as classes of ordered structures of some vocabulary rather than languages over some alphabet. Consequently, parameterized problems are considered as subsets $Q \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ for some vocabulary τ. It is required that for each $k \in \mathbb{N}$ the kth slice $Q_k := \{\mathcal{A} \in \mathrm{ORD}[\tau] \mid (\mathcal{A}, k) \in Q\}$ of Q is closed under isomorphisms.

We recall some definitions from [14]. Let L be a logic. A parameterized problem $Q \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ is *slicewise* L-*definable*, if there is a computable function $\delta : \mathbb{N} \to \mathrm{L}$ such that for all $\mathcal{A} \in \mathrm{ORD}[\tau]$ and $k \in \mathbb{N}$ we have

$$(\mathcal{A}, k) \in P \quad \iff \quad \mathcal{A} \models \delta(k).$$

A family $(\mathrm{L}_s)_{s\in\mathbb{N}}$ of logics *captures* a parameterized complexity class C if for every vocabulary τ and every parameterized problem $Q \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ we have

$$Q \in \mathrm{C} \iff \text{there is an } s \geq 1 \text{ such that } Q \text{ is slicewise } \mathrm{L}_s\text{-definable.}$$

If this is the case, we write

$$\mathrm{C} = \bigcup_{s\geq 1} \text{slicewise-L}_s.$$

For $s \geq 1$, let $\mathrm{LFP}^{[s]}$ consist of all formulas of least fixed-point logic of the form

$$[\mathrm{LFP}_{\bar{x},X}\varphi]\bar{z},$$

where X is of arity $\leq s$ and $\varphi \in \mathrm{FO}^{[s]}$, that is, φ is a first-order formula, and in φ at most s individual variables are quantified (note that in φ a variable may be quantified several times and that, besides the quantified variables, φ may contain other variables; thus, $\mathrm{FO}^{[s]}$ strictly contains the finite variable fragment FO^s consisting of the first-order formulas with at most s variables at all). Finally, $\Sigma_1\mathrm{LFP}^{[s]}$ denotes the class of all formulas of the form $\exists x_1 \ldots \exists x_\ell \varphi$, where $\varphi \in \mathrm{LFP}^{[s]}$, and $\Pi\,\mathrm{LFP}^{[s]}$ the class of formulas $Q_1 x_1 \ldots Q_\ell x_\ell \varphi$, where $\ell \geq 1$, $Q_1, \ldots, Q_\ell \in \{\forall, \exists\}$, and $\varphi \in \mathrm{LFP}^{[s]}$.

The proof of the following theorem parallels that of Theorem 7.1 in [13]. We use the following two facts, the first implicit in [24] and the second in the proof of the Immerman-Vardi Theorem [17, 23] (also see [11]). Fix $s \geq 1$:

- $(*)$ There is an algorithm that for every vocabulary τ, every formula $\varphi(\bar{y}) \in \mathrm{LFP}^{[s]}$ of vocabulary τ, every τ-structure $\mathcal{A}$, and every tuple $\bar{a} \in A^{\mathrm{length}(\bar{y})}$ decides if $\mathcal{A} \models \varphi(\bar{a})$ in time $O(\|\varphi\| \cdot \|(\mathcal{A}, \bar{a})\|^{2s})$.
- $(**)$ For every vocabulary τ there is a $t \in \mathbb{N}$ and a computable function that associates with every $d \in \mathbb{N}$ and every $O(n^s)$-algorithm accepting a class C of structures $(\mathcal{A}, \bar{a})$ with $\mathcal{A} \in \mathrm{ORD}[\tau]$ and $\bar{a} \in A^d$ a formula $\varphi(\bar{y}) \in \mathrm{LFP}^{[t]}$ of vocabulary τ such that for every τ-structure $\mathcal{A}$ and every $\bar{a} \in A^d$ we have $\mathcal{A} \models \varphi(\bar{a})$ if and only if $(\mathcal{A}, \bar{a}) \in C$.

Theorem 31.

$$\mathrm{W[P]} = \bigcup_{s\geq 1} \text{slicewise-}\Sigma_1\mathrm{LFP}^{[s]}.$$

Proof: First, suppose that $Q \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ is slicewise $\Sigma_1\mathrm{LFP}^{[s]}$-definable via the computable function $\delta : \mathbb{N} \to \Sigma_1\mathrm{LFP}^{[s]}$. Let $\delta(k) := \exists\bar{y}\varphi(\bar{y})$ and $\bar{y} = y_1 \ldots y_d$. Let $\mathbb{A}_{\varphi(\bar{y})}$ be the algorithm obtained by $(*)$ by fixing the input formula $\varphi(\bar{y})$. Then,

$$\begin{aligned}
(\mathcal{A}, k) \in Q \quad &\iff \quad \mathcal{A} \models \exists\bar{y}\varphi(\bar{y}) \\
&\iff \quad \text{there are } a_1, \ldots, a_d \in A \text{ with } \mathcal{A} \models \varphi(\bar{a}) \\
&\iff \quad \text{there are } a_1, \ldots, a_d \in A \text{ such that } \mathbb{A}_{\varphi(\bar{y})} \text{ accepts } (\mathcal{A}, \bar{a}).
\end{aligned}$$

Hence, a nondeterministic Turing machine M carrying out the Algorithm 1 accepts Q. Therefore $Q \in$ W[P], since M is a machine satisfying the requirements in Corollary 11: The number of steps needed for

1 compute $\delta(k) = \exists y_1 \ldots \exists y_d \varphi(\bar{y}) \in \Sigma_1\text{LFP}^{[s]}$
2 compute $\mathbb{A}_{\varphi(\bar{y})}$ (cf. $(*)$)
3 guess elements $a_1, \ldots, a_d$ in A
4 simulate $\mathbb{A}_{\varphi(\bar{y})}$ on input $(\mathcal{A}, \bar{a})$
5 **if** $\mathbb{A}_{\delta(k)}$ accepts $(\mathcal{A}, \bar{a})$
6 **then** accept
7 **else** reject.

Algorithm 1

lines 1 and 2 only depends on k; line 3 consists in guessing $d \cdot \log |A|$ ($\leq g(k) \cdot \log \|\mathcal{A}\|$) bits; and (compare line 4) $\mathbb{A}_{\varphi(\bar{y})}$, on input $(\mathcal{A}, \bar{a})$ performs $O(\|(\mathcal{A}, \bar{a})\|^{2s}) = h(k) \cdot O(\|\mathcal{A}\|^{2s+1})$ steps for some computable h.

For the other direction, suppose that $Q \subseteq \text{ORD}[\tau] \times \mathbb{N}$ is in W[P]. Choose a nondeterministic Turing machine M accepting Q according to Corollary 11 (or Lemma 7): For every $(\mathcal{A}, k) \in Q$, there is a run of M accepting $(\mathcal{A}, k)$ of length at most $g(k) \cdot n^s$ (for some computable g and $s \geq 1$) such that the nondeterministic steps are among the first $g(k) \cdot \log\ n$ ones and that they consist in choosing a 0–1 string of length $g(k) \cdot \log\ n$. Note that for some c depending on τ only, we have for any structure $\mathcal{A}$, $(\log\ n =) \log \|\mathcal{A}\| \leq c \cdot \log |A|$. Hence, for any instance $(\mathcal{A}, k)$ of Q, we can arrange such an 0–1 string in $f(k)$ ($\leq c \cdot g(k)$) blocks of $\log |A|$ bits, each such block corresponding to the binary representation of a number $\leq |A|$ and hence, to an element of A. Let $a_1, \ldots, a_{f(k)}$ be the corresponding elements. Now, we can view the deterministic part of M as a $g(k) \cdot n^s$ bounded algorithm applied to $(\mathcal{A}, \bar{a})$. Then $(**)$ yields, for every k, an $\text{LFP}^{[t]}$-formula $\varphi(\bar{y})$ such that for all $\mathcal{A} \in \text{ORD}[\tau]$

$$\begin{aligned} \mathcal{A} \models \exists \bar{y} \varphi(\bar{y}) \quad &\iff \quad M \text{ accepts } (\mathcal{A}, k) \\ &\iff \quad (\mathcal{A}, k) \in Q. \end{aligned}$$

This gives the desired slicewise definition of Q in $\Sigma_1\text{LFP}^{[t]}$. □

Essentially the same proof also gives a characterisation of AW[P]:

Theorem 32.

$$\text{AW[P]} = \bigcup_{s \geq 1} \text{slicewise-}\Pi\,\text{LFP}^{[s]}.$$

Besides the various weighted satisfiability problems that are used to define the W and AW classes, there is another generic family of problems that are complete for these classes: model-checking problems. The *parameterized model-checking problem* for a logic L is the following problem:

p-MC(L)
Input: A structure $\mathcal{A}$ and an L-sentence φ.
Parameter: $\|\varphi\|$.
Problem: Decide if $\mathcal{A} \models \varphi$.

The parameterized model-checking problem for first-order logic is complete for the class AW$[*]$ under FPT-reductions [10]. For each of the classes of the W-hierarchy and the A-hierarchy there is a fragment of first-order logic whose parameterized model checking problem is complete for the class [8, 13]. Using Theorems 31 and 32 we get the following:

Theorem 33. *(1) For all $s \geq 1$ the parameterized model-checking problem for $\Sigma_1\text{LFP}^{[s]}$ is complete for* W[P] *under FPT-reductions.*

(2) For all $s \geq 1$ the parameterized model-checking problem for $\Pi\,\mathrm{LFP}^{[s]}$ is complete for AW[P] *under FPT-reductions.*

Proof: (1) Note that the algorithm existing according to $(*)$ works for all vocabularies τ; hence, similarly as in the first part of the preceding proof, we get p-MC($\Sigma_1\mathrm{LFP}^{[s]}$) $\in$ W[P] for all $s \geq 1$. By Theorem 31, the W[P]-complete (ordered) problem p-CIRCUIT is slicewise-$\Sigma_1\mathrm{LFP}^{[s_0]}$ for some $s_0 \geq 1$. Thus, p-MC($\Sigma_1\mathrm{LFP}^{[s]}$) is W[P]-hard for $s \geq s_0$. We shall see in the proof of Theorem 35 that we can choose $s_0 = 1$.

The proof of (2) is similar. □

Remark 34. The only standard parameterized complexity classes for which we have not been able to give a descriptive characterisation in the spirit of Theorems 31 and 32 are W[SAT] and AW[SAT]. We neither know complete parameterized model-checking problems for these logics.

However, Papadimitriou and Yannakakis [21] showed that differently parameterized model-checking problems are complete for these classes. Instead of parameterizing the model-checking by the length of the input formula, they parameterized it by the number of variables:

p-MC(L, var)	
Input:	A structure $\mathcal{A}$ and an L-sentence φ.
Parameter:	Number of variables in φ.
Problem:	Decide if $\mathcal{A} \models \varphi$.

Essentially, they proved that p-MC(FO, var) is complete for AW[SAT] and p-MC(Σ_1, var) is complete for W[SAT], where Σ_1 is the set of all existential first-order formulas in prenex normal form. (We give the precise statement for W[SAT]: Let $r \geq 1$ and denote by $\Sigma_1[r]$ the set of Σ_1-formulas without function symbols and where all relation symbols are of arity $\leq r$. Then p-MC($\Sigma_1[r]$, var) is complete for W[SAT].)

5.3. Fagin Definability. Besides the model-checking problems there is another class of natural parameterized problems derived from logic, which has been dubbed Fagin-definable problems in [13].

Let ψ be a formula of vocabulary $\tau \cup \{Z\}$, where Z is an r-ary relation symbol not contained in τ. It *Fagin-defines* a parameterized problem p-FD$_{\psi(Z)}$:

p-FD$_{\psi(Z)}$	
Input:	A τ-structure $\mathcal{A}$.
Parameter:	$k \in \mathbb{N}$.
Problem:	Is there $B \subseteq A^r$ with $\lvert B\rvert = k$ and $(\mathcal{A}, B) \models \psi$?

For a logic L, we let FD(L) denote the class of all problems that are Fagin defined by a formula in L. Recall that for a class C of parameterized problems, $[\mathrm{C}]^{\mathrm{FPT}}$ denotes the closure of this class under FPT-reductions. Downey, Fellows, and Regan [8] proved that for each $t \geq 1$

$$\big[\mathrm{FD}(\Pi_t)\big]^{\mathrm{FPT}} = \mathrm{W}[t],$$

where Π_t is the set of all first-order formulas of the form

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

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

We give a characterisation of the class W[P] in terms of Fagin definability:

Theorem 35.

$$\Big[\mathrm{FD}\big(\mathrm{FO}(\mathrm{LFP})\big)\Big]^{\mathrm{FPT}} = \mathrm{W}[\mathrm{P}],$$

More precisely, there is an FO(LFP)*-formula ψ such that* p-FD$_{\psi(Z)}$ *is complete for* W[P] *under* FPT*-reductions, and for each* FO(LFP)*-formula φ the problem* p-FD$_{\varphi(Z)}$ *is contained in* W[P].

Proof: We first prove that there is an FO(LFP)-formula ψ such that p-FD$_{\psi(Z)}$ is hard for W[P] under FPT-reductions.

Recall Example 30, where it was shown how to define the monotone circuit value problem in FO(LFP). We can easily extend this to the circuit value problem for circuits in *negation normal form*, that is, circuits in which negations gates only appear directly above input gates. To do this, we simply have to add a clause to the formula $\varphi(x)$ of Example 30 which says that if x is a negation gate then it evaluates to TRUE if it has a child not contained in the set $T^{\mathcal{C}}$ of input nodes set to TRUE. We get a formula

$$\varphi'(x) = Tx \vee \big(G_{\neg}x \wedge \exists y(Exy \wedge \neg Ty)\big) \vee \big(G_{\bigwedge}x \wedge \forall y(Exy \to Xy)\big) \vee \big(G_{\bigvee}x \wedge \exists y(Exy \wedge Xy)\big),$$

and let $\psi' = [\text{LFP}_{x,X}\varphi'](\text{out})$. Then a circuit $\mathcal{C}$ in negation normal form evaluates to TRUE under the assignment that sets precisely the input nodes in $T^{\mathcal{C}} \subseteq I^{\mathcal{C}}$ to true if, and only if,

$$(\mathcal{C}, T^{\mathcal{C}}) \models \psi'.$$

Thus a circuit $\mathcal{C}$ in negation normal form has a satisfying assignment of weight k if, and only if, there exists a k-element set $T^{\mathcal{C}} \subseteq I^{\mathcal{C}}$ such that $(\mathcal{C}, T^{\mathcal{C}}) \models \psi'$. Letting $\psi = \psi' \wedge \forall x(Tx \to Ix)$, we see that, on the class of circuits in negation normal form, p-FD$_{\psi(T)}$ is precisely the weighted satisfiability problem.

Since every circuit can be transferred to an equivalent circuit in negation normal form in polynomial time, this yields an FPT-reduction from WSAT(CIRCUIT) to p-FD$_{\psi(T)}$ and thus shows that p-FD$_{\psi(T)}$ is W[P]-hard under FPT-reductions.

It remains to prove that p-FD$_{\psi(Z)}$ is in W[P] for every $\psi \in$ FO(LFP). By using a well-known normal form for least fixed-point logic [17] (also see [11]) we can assume that ψ has the form

$$\exists y[\text{LFP}_{\bar{x},X}\chi(\bar{x}, y)](y \dots y),$$

where χ is a first-order formula of vocabulary $\tau \cup \{X, Z\}$ and neither X nor Z are contained in τ. For notational simplicity, we assume that X is unary. Let s be the number of variables quantified in χ. We show that p-FD$_{\psi(X)}$ is slicewise $\Sigma_1\text{LFP}^{[s]}$-definable. Given a parameter k, set (with new variables $z_1, \dots, z_k$)

$$\varphi^k := \exists z_1 \dots \exists z_k \exists y[\text{LFP}_{x,X}(\bigwedge_{1 \le i < j \le k} z_i \neq z_j \wedge \chi^k)](y \dots y),$$

where χ^k is obtained from χ by replacing each atomic formula of the form Zt by $(t = z_1 \vee \dots \vee t = z_k)$. Then $\varphi^k \in \Sigma_1\text{LFP}^{[s]}$ and for every structure $\mathcal{A}$,

$$(\mathcal{A}, k) \in \text{p-FD}_{\psi(Z)} \iff \mathcal{A} \models \varphi^k.$$

This shows that p-FD$_{\psi(Z)}$ is slicewise $\Sigma_1\text{LFP}^{[s]}$-definable and thus in W[P] by Theorem 31. □

6. Conclusions

By giving machine characterisations and logical descriptions of the classes W[P] and AW[P] we feel that we have gained a much clearer understanding of these classes. The logical descriptions place the classes into a uniform framework that we had started to develop in earlier work, so that now we have a fairly comprehensive picture of the logical side of parameterized complexity classes. The only important classes not yet integrated into this picture are the classes W[SAT] and AW[SAT].

The machine characterisation of W[P] is very simple and natural and provides a precise connection between parameterized complexity theory and limited nondeterminism. Moreover, as far as we know it is the first machine characterisation for any of the standard intractable parameterized complexity classes; only characterisations via complete problems were known before. We gave similar characterisations for the classes W[1], A[t] for $t \geq 1$, AW[$*$], and AW[t]. Curiously, we were *not* able to give such characterisations for the classes W[t] for $t \geq 2$. It remains an interesting open problem to find natural machine characterisations for these classes.

Another open problem is the relation between the classes AW[P] and [uniform-XNL]$^{\text{FPT}}$, the closure of uniform-XNL under FPT-reductions. This is equivalent to the questions of whether the problem CNTMC is contained in AW[P] and whether it is hard for AW[P]. We conjecture that the answer to both of these questions is negative.

Appendix A provides an overview over all classes studied in this paper and a few more.

References

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

[2] M. Alekhnovich and A. Razborov. Resolution is not automatizable unless W[P] is tractable. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, 2001. To appear.

[3] 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.

[4] L. Cai, J. Chen, R.G. Downey, and M.R. Fellows. On the parameterized complexity of short computation and factorization. *Archive for Mathematical Logic*, 36:321–337, 1997.

[5] M. Cesati. The Turing way to the parameterized intractability, 2001. Submitted for publication.

[6] A.K. Chandra, D. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

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

[8] 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.

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

[10] 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.

[11] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 2nd edition, 1999.

[12] R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974.

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

[14] J. Flum and M. Grohe. Describing parameterized complexity classes. In H. Alt and A. Ferreira, editors, *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2285 of *Lecture Notes in Computer Science*, pages 359–371. Springer-Verlag, 2002.

[15] 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.

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

[17] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[18] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16:760–778, 1987.

[19] N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1999.

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

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

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

[23] M.Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 137–146, 1982.

[24] M.Y. Vardi. On the complexity of bounded-variable queries. In *Proceedings of the 14th ACM Symposium on Principles of Database Systems*, pages 266–276, 1995.

Appendix A: Overview

The following figure contains the most important parameterized complexity classes. Arrows indicate inclusion.