Parameterized Complexity of Fixed-Variable Logics

Christoph Berkholz[1] **and Michael Elberfeld**[2]

1 **RWTH Aachen University, Aachen, Germany**
`berkholz@cs.rwth-aachen.de`
2 **RWTH Aachen University, Aachen, Germany**
`elberfeld@cs.rwth-aachen.de`

Abstract

We study the complexity of model checking formulas in first-order logic parameterized by the number of distinct variables in the formula. This problem, which is not known to be fixed-parameter tractable, resisted to be properly classified in the context of parameterized complexity. We show that it is complete for a newly-defined complexity class that we propose as an analog of the classical class PSPACE in parameterized complexity. We support this intuition by the following findings: First, the proposed class admits a definition in terms of alternating Turing machines in a similar way as PSPACE can be defined in terms of polynomial-time alternating machines. Second, we show that parameterized versions of other PSPACE-complete problems, like winning certain pebble games and finding restricted resolution refutations, are complete for this class.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Parameterized complexity, polynomial space, first-order logic, pebble games, regular resolutions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.109

1 Introduction

The main goal of computational complexity theory is to distinguish between tractable and intractable problems. In classical complexity theory tractable problems are those that can be solved in polynomial time, whereas intractable problems require exponential time (most notably NP-complete problems, but also problems complete for higher levels of the polynomial hierarchy, PSPACE, and EXPTIME). In parameterized complexity theory, tractable problems are in FPT and can be solved in time $f(k) \cdot n^{O(1)}$, whereas intractable problems require a running time of $n^{f(k)}$.

Beside distinguishing between just tractable and intractable problems, looking at different levels of intractability for NP-hard problems (by comparing them with respect to polynomial-time reductions) led to understanding the importance of the polynomial hierarchy as well as polynomial-space, and exponential-time computations. Already during the incubation of parameterized complexity theory different levels of parameterized intractability were observed based on comparing problems with respect to fixed-parameter tractable reduction (fpt-reductions). This turned into a flourishing research area where classes that were initially defined in an adhoc way by considering yet unclassified problems and their closures under fpt-reductions turned out to be definable using descriptive characterizations in terms of first-order logic.

Paper's Issue. The class XP of parameterized problems that can be solved in time $n^{f(k)}$ is the analog of EXPTIME in parameterized complexity as both classes contain all problems of intractable running time. Indeed, it turned out that parameterizations of EXPTIME-complete problems tend to be complete for XP [2, 3, 4]. Several classes are proposed as parameterized analogs of PSPACE. This includes the classes AW[SAT] and AW[*], which both admit alternative characterizations in terms of model checking first-order formulas, as well as AW[P], which can be defined based on deterministic Turing machines that access a certificate containing blocks of existential (nondeterministic) bits and blocks of universal (nondeterministic) bits. For all of these PSPACE-analogs the alternation used to solve problems is bounded by a function in terms of the parameter. On the one side, the classes defined in this way helped to classify parameterized versions of PSPACE-complete problems where the parameter is used to bound the use of alternation. On the other side, problems that result from more general parameterizations of PSPACE-complete problems resisted to be classified using these classes. A prominent example is the problem of evaluating first-order formulas. It is PSPACE-complete [18] in the classical setting, and known to be in XP when parameterized by the number of distinct variables in the formula, but not known to be hard for this class. The importance of the fixed variable fragments of first-order logic stems from the fact that k-variable formulas can be evaluated in time $n^{O(k)}$. In addition, by reusing the variables one has access to an unlimited number of quantifier alternations, which makes this fragment much more expressive than fragments with bound quantifier depth. A similar observation can be made for determining the winner in a classical acyclic pebble game and finding linear depth resolution refutations of bounded width; their unparameterized versions are shown to be complete for PSPACE in [14] and [3], respectively, but they are not complete for known parameterized analogs of PSPACE as they require unbounded alternation.

Paper's Contributions. We properly classify the parameterized complexity of these problems by presenting the following contributions during the course of the present paper: (1) We consider the closure under fpt-reductions of model checking first-order formulas parameterized by the number of distinct variables in the formula and sort this class into the hierarchy of known levels of parameterized intractability. (2) We prove that the newly defined class SXP, which stands for shallow XP, has a natural characterization in terms of alternating Turing machines (with unbounded alternation) in a similar way as PSPACE can be characterized in terms of alternating polynomial time. For this result, we apply techniques from descriptive complexity theory [13] to simulate the behavior of alternating machines using first-order formulas. (3) We show that other PSPACE-complete problems are complete for this class under fpt-reductions when parameterized in a natural and very general way. We first simulate the model checking game for k-variable logic within the acyclic $k+2$-pebble game of Kasai, Adachi and Iwata, which was introduced to simulate PSPACE machines, to show that the pebble game is complete for our new class when parameterized by the number of pebbles. Afterwards, we use a known reduction from the acyclic pebble game to regular resolution of bounded width to show that finding resolution refutations of linear depth and width k is another PSPACE-complete problem that fits in our parameterized analog of PSPACE when parameterized by the width. Interestingly, the pebble game and bounded width resolution have more general versions that are EXPTIME-complete, classically, and XP-complete, parameterized.

Paper's Organization. The next section defines concepts and terminology related to parameterized complexity and first-order logic. The subsequent Sections 3, 4, 5 present, respectively, the definition of our newly proposed parameterized analog of PSPACE, a machine characterization for the class, and complete problems.

2 Background

The present section provides background from parameterized complexity and first-order logic as well as establishes notation related to parameterized versions of model checking first-order formulas. The used definitions and notations closely follow the book of Flum and Grohe [12]; see also this book for standard results in parameterized complexity mentioned below.

Parameterized complexity. A *parameterized problem* is a pair (P, κ) consisting of a *(classical) problem* $P \subseteq \{0,1\}^*$ and a *parameterization* $\kappa\colon \{0,1\}^* \to \{1\}^*$ that is polynomial-time computable; we commonly denote it by p-κ-P. Given an *instance* $x \in \{0,1\}^*$, we use the shorthands $n := |x|$, its *size*, and $k := |\kappa(x)|$, its *parameter*. We denote by FPT the class of parameterized problems (P, κ) that are solvable by a deterministic Turing machine (DTM) whose runtime is at most $f(k) \cdot n^{O(1)}$ for a computable function $f\colon \mathbb{N} \to \mathbb{N}$; (parameterized) problems in FPT are *fixed-parameter tractable*. The class XP is defined like FPT, but using time bounds $n^{f(k)}$. The deterministic time hierarchy theorem implies that FPT is a proper subclass of XP.

An fpt-*reduction* from a parameterized problem (P,κ) to a parameterized problem (P',κ') is a mapping $r\colon \{0,1\}^* \to \{0,1\}^*$ that is computable by a DTM in time $f(k) \cdot n^{O(1)}$ for a computable function $f\colon \mathbb{N} \to \mathbb{N}$, such that for every $x \in \{0,1\}^*$, we have (1) $x \in P$ if, and only if, $r(x) \in P'$, and (2) $|\kappa'(r(x))| \leq |g(\kappa(x))|$ for some reduction-dependent function $g\colon \{1\}^* \to \{1\}^*$. Given a parameterized problem (P', κ'), the *closure of* (P', κ') *under* fpt-*reductions*, denoted by $[(P', \kappa')]^{\mathrm{fpt}}$, is the class of all problems (P, κ) with an fpt-reduction from (P, κ) to (P', κ'). Later we study problems that are complete for XP or other complexity classes of parameterized problems between FPT and XP. In all of these cases, completeness is defined with respect to fpt-reductions.

First-order logic. We start to define the syntax and semantics of first-order logic: A *vocabulary* τ is a nonempty and finite set of *relation symbols* R_i with *arities* $\mathrm{arity}(R_i) \in \mathbb{N}$. A *structure* $\mathcal{A}$ over τ consists of a finite set A, its *universe*, and a *relation* $R_i^{\mathcal{A}} \subseteq A^{\mathrm{arity}(R_i)}$ for every relation symbol R_i of τ. Based on *(element) variables* x_i, $i \in \mathbb{N}$, *first-order formulas* (FO-formulas) over a vocabulary τ are (1) *atomic* formulas $x_i = x_j$ and $(x_1, \ldots, x_r) \in R_i$, and (2) *composed* formulas $\neg\varphi$, $\varphi \wedge \psi$ and $\varphi \vee \psi$, which are based on *connectives*, and $\exists x_i\, \varphi$ and $\forall x_i\, \varphi$, which are based on *quantifiers*. Given a structure $\mathcal{A}$ and a formula φ over the same vocabulary τ, $\mathcal{A}$ *satisfies* φ if the usual model relation $\mathcal{A} \models \varphi$ holds. Two formulas φ and ψ are *equivalent* if they are satisfied by exactly the same structures.

In order to define parameterized problems and complexity classes that are based on first-order formulas, we define classes of formulas and parameters of formulas: First of all, we only consider FO-formulas in *negation normal form*, whose $\neg$-connectives are immediately in front of atomic formulas. This does not restrict the set of FO-formulas in the sense that every FO-formula can be turned into an equivalent formula in negation normal form by recursively applying the rules of De Morgan. We denote this set of formulas by FO. The *quantifier alternation depth* of a formula φ, denoted by $\mathrm{qad}(\varphi)$, is the number of alternations from $\exists$- to $\forall$-quantifiers on any root-to-leaf path in φ's syntax tree plus 1; and plus 2 if the first quantifier is $\forall$. For every $t \in \mathbb{N}$, the class of formulas $\varphi \in$ FO with $\mathrm{qad}(\varphi) = t$ is ALT_t. A formula φ is in *prenex normal form* if $\varphi = Q_1 x_1 \ldots Q_\ell x_\ell\, \psi$ where the Q_i are quantifiers and

ψ does not contain quantifiers. The class of all these formulas is PRENEX and, for every $t \in \mathbb{N}$, we set $\Sigma_t := \text{PRENEX} \cap \text{ALT}_t$. Let $\varphi \in \text{FO}$. We denote the number of distinct variables of φ by $\text{var}(\varphi)$. The size of φ, denoted by $\text{size}(\varphi)$, is the total number of symbols used to write down φ.

Model checking formulas. In order to consider vocabularies, structures, and formulas as part of instances to computational problems, we encode them using binary strings in a standard way as done in [12]. We write $\text{enc}(\tau)$ for the binary string encoding of a vocabulary τ; $\text{enc}(\mathcal{A})$ and $\text{enc}(\varphi)$ are used for the encoding of a structure $\mathcal{A}$ and a formula φ, respectively. Given a class of first-order formulas $\mathcal{C} \subseteq \text{FO}$, we denote by $\text{MC}(C)$ the *model checking problem* for formulas from C: A positive instance of $\text{MC}(C)$ consists of an encoded vocabulary $\text{enc}(\tau)$ as well as an encoded structure $\text{enc}(\mathcal{A})$ and an encoded formula $\text{enc}(\varphi)$ with (1) $\varphi \in C$, (2) $\mathcal{A}$ and φ are both over τ, and (3) $\mathcal{A} \models \varphi$. We consider parameterized versions of model checking problems with respect to several classes of formulas in the following section, and their closure under fpt-reductions.

3 Parameterized complexity of first-order logic

Previous studies (mainly subsumed by the book [12]) observed that the parameterized complexity of model checking first-order formulas with respect to several classes of formulas and parameterizations is not only interesting from the perspective of solving the problem, but as a central concept to study parameterized intractability, itself. That means, model checking formulas in FO-logic is commonly used to define levels of parameterized intractability, which make up a candidate hierarchy of complexity classes between the, provably distinct, classes FPT and XP.

A first family of classes are defined as *analogs of* NP in the parameterized setting: This spans the W-hierarchy, with its frequently used first level $\text{W}[1] := [\text{p-size-MC}(\Sigma_1)]^{\text{fpt}}$ [7, 9, 11], as well as $\text{W[SAT]} := [\text{p-var-MC}(\Sigma_1)]^{\text{fpt}}$ [7, 16] and the class W[P], whose definition equals the one of FPT except that the used deterministic Turing machines have access to a nondeterministic certificate of $f(k) \cdot \log n$ nondeterministic bits [7, 6]. These classes are considered to be analogs of NP in the parameterized world since it is possible to nondeterministically guess candidate solutions to, say, graph problems of a parameter-bounded size, and verify the guessed solution. The computational power we have for the verifying step depends on the particular class. Since we consider FPT as the lowest level, of tractable problems, the classes are defined by taking the closure of these problems under fpt-reductions.

In a similar fashion, parameterized *analogs of the polynomial-time hierarchy (*PH*)* are defined based on formulas with constant alternations with the most prominent suggestion being the A-hierarchy with levels $A[t] := [\text{p-size-MC}(\Sigma_t)]^{\text{fpt}}$ for $t \in \mathbb{N}$ [10, 11].

A third kind of classes are designed to be *analogs of* PSPACE in the parameterized setting: In this case, not only existential, but also universal, nondeterminism is permitted; still nondeterminism is bounded in terms of the parameter. The most powerful of these classes is AW[P], which is defined as W[P], but this time the acceptance behavior of the DTM depends on a length-$(f(k) \cdot \log n)$ certificate containing both existential and universal nondeterministic bits where the number of alternations is bounded by $f(k)$. Classes that are defined via a less powerful solution verification phase are $\text{AW[SAT]} := [\text{p-var-MC}(\text{PRENEX})]^{\text{fpt}}$ and $\text{AW}[*] := [\text{p-size-MC}(\text{PRENEX})]^{\text{fpt}}$. All of the mentioned PSPACE-analogs are originally defined in terms of Boolean circuits and propositional logic [1]; the definitions based on machines and model checking first-order formulas are taken from [12, Chapter 8].

While the parameterized approaches towards mirroring the behavior of PSPACE have proven to be useful to classify a large number of problems, some problems remained unclassified and, hence, not well understood. These are parameterized problems whose solutions are based on existential and universally nondeterministic guesses, but where it seems not possible to bound the nondeterministic guesses in terms of the parameter. Thus, the parameter seems to play a different role for these problems. A prominent example of such a problem is model checking first-order formulas that only use a fixed (parameter-bounded) number of distinct variables. As the variables can be reused, the quantifier alternation does not depend on the parameter. This observation leads us to defining the following *parameterized analog* of PSPACE with unbounded alternations. In Section 4 we realize that the defined complexity class has a characterization in terms of alternating Turing machines that is similar to XP, but using shallow parallel computations; hence, we choose the name SXP with S standing for "shallow".

▶ **Definition 1.** SXP := $[\text{p-var-MC}(\text{FO})]^{\text{fpt}}$

The class SXP is contained in XP and contains AW[P]. Figure 1 shows the relations between these and other classes from parameterized complexity mentioned above.

▶ **Fact 2.** SXP $\subseteq$ XP

▶ **Lemma 3.** AW[P] $\subseteq$ SXP

Proof Sketch. Let $(P, \kappa) \in$ AW[P] via a DTM M running in time $f(k) \cdot n^c$ for some computable function f and constant c, and using nondeterministic certificates of length $f(k) \cdot \log n$ of existential and universal bits; with $f(k)$ alternations. The first phase of our fpt-reduction to p-var-MC(FO) reduces the problem of simulating the computation of M on length-n and parameter-k instances to the circuit evaluation problem as described in [15] for the classical class P. The second step replaces the task of evaluating the circuit by model checking a constant-variable first-order formula that defines the evaluation problem for circuits (the same approach is commonly used to show that model checking first-order formulas with just 2 variables is complete for P). To also take the alternating certificate into account, which is given to the DTM, our formula is enriched by existential quantifiers, which guess existential bits, and universal quantifiers, which guess universal bits; this construction uses a single quantifier to handle a length-($\log n$) substring of the certificate by first applying the "$k \cdot \log n$"-trick (see [12, Corollary 3.13] for details) to the circuit. Both the circuit and the formula can be constructed using an fpt-reduction. The number of variables used by the formula is bounded by a function in the original parameter. ◀

4 Alternative Characterizations

The classes that are defined via model checking first-order formulas in the previous section are all defined by first taking a prototypical model checking problem for first-order formulas of a restricted syntax along with a parameter, which is the formulas size or number of distinct variables. Then the closures of these problems under fpt-reductions are considered, which captures certain kinds of parameterized intractable problems. The prototypical problems are chosen in order to mirror the behavior of a classical class in the parameterized setting. In this section we present an alternative characterization of SXP in terms of alternating Turing machines. It shows how the behavior of the class SXP (of parameterized problems) parallels the behavior of the class PSPACE (of classical problems).

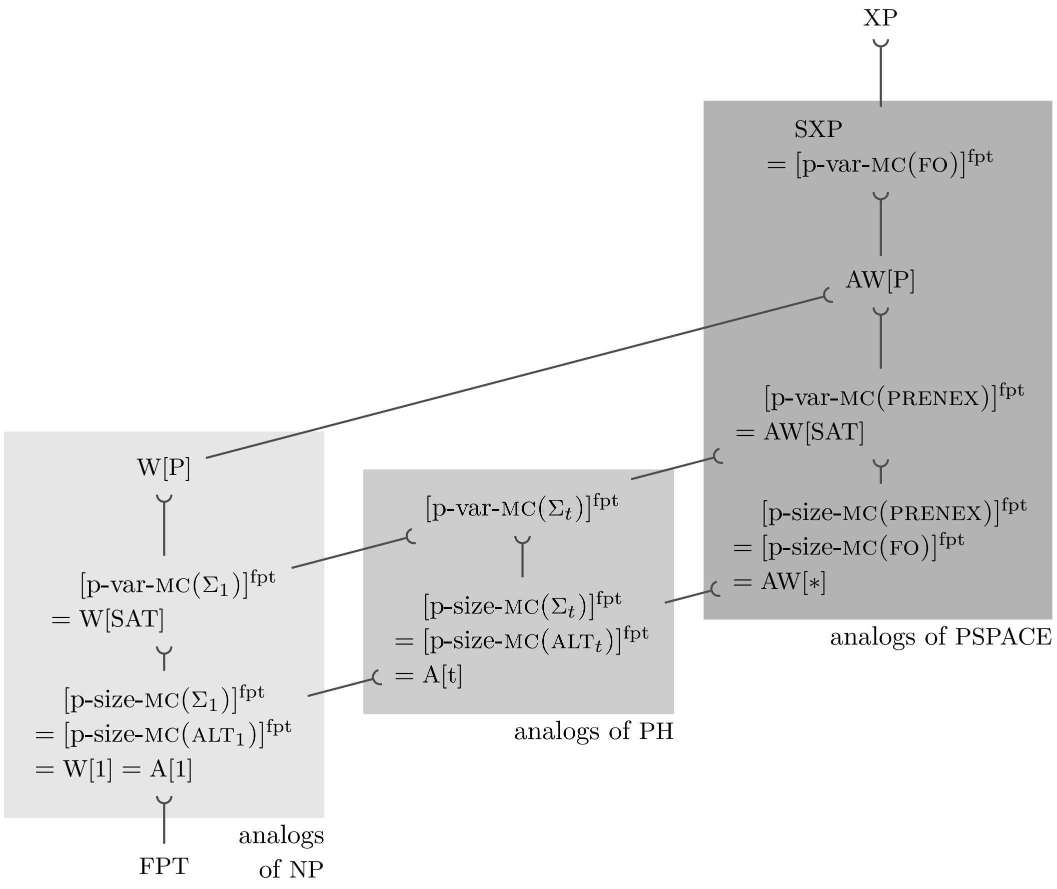

Figure 1 We have the following inclusions of parameterized complexity classes for every $t \in \mathbb{N}$; where C —(D indicates C $\subseteq$ D for classes C and D. While FPT is commonly considered to be the analog of P in parameterized complexity and XP is an analog of EXPTIME, there are several suggestion to reflect the behavior of the classical classes NP, the levels of PH, and PSPACE. Our suggestion for a parameterized version of PSPACE is based on parameterizing first-order model checking via the number of distinct variables in formulas.

An *alternating Turing machine* (ATM) M consists of a set of *states* Q that is partitioned into a set of *existential states* $Q_\exists$ and a set of *universal states* $Q_\forall$. Its *(nondeterministic) transitions* are encoded by a relation $\Delta \subseteq Q \times \Sigma^k \times Q \times \Sigma^k \times \{\texttt{LEFT}, \texttt{RIGHT}\}^k$ where $(q, \sigma_1, \ldots, \sigma_k, q', \sigma'_1, \ldots, \sigma'_k, d_1, \ldots, d_k) \in \Delta$ means that if M is in state q and reads the symbol σ_i on tape i, for $i \in \{1, \ldots, k\}$, then it can write the symbol σ'_i on tape i, for $i \in \{1, \ldots, k\}$, and moves its heads as defined by the d_i. We consider only machines M that halt on every computation path. The acceptance behavior of an ATM is defined recursively (without using accepting and rejecting states explicitly) as follows: A *universal* configuration *accepts* if every immediate successor configuration accepts, an *existential* configuration *accepts* if there exists an immediate successor configuration that accepts. M accepts an input if the *starting configuration* accepts.

From the proof of the well known characterization from Chandra et al. [5] of polynomial deterministic time in terms of alternating logarithmic space, we get the following alternative definition of XP. This parallels the definition of EXPTIME in terms of ATMs using polynomial space.

▶ **Fact 4.** XP *is the class of parameterized problems* (P, κ) *that are accepted by* ATMs *using space at most* $f(k) \cdot \log n$.

While for EXPTIME problems we do not hope to lower the run-time substantially by using alternation, alternation speeds-up the solution of problems in PSPACE since it equals the class of problems accepted by ATMs in polynomial time [5]. The following lemma states that our proposed parameterized version of PSPACE has a similar behavior. Its problems can be solved by ATMs using $f(k) \cdot \log n$ space, but only running in $f(k) \cdot n^{O(1)}$ time. The proof of the lemma is based on a refined view on the PSPACE-completeness of model checking first-order formulas [17, 18] as well as ideas from descriptive complexity theory [13].

▶ **Theorem 5.** SXP *is the class of parameterized problems* (P, κ) *that are accepted by* ATMs *using space at most* $f(k) \cdot \log n$ *and running in time at most* $f(k) \cdot n^{O(1)}$.

Proof. We start to show how problems in SXP, which are fpt-reducible to p-var-MC(FO), can be solved by $(f(k) \cdot \log n)$-space- and $f(k) \cdot n^{O(1)}$-time-bounded ATMs. Classical results from Chandra et al. [5] imply that FPT is the class of parameterized problems (P, κ) accepted by ATMs using space at most $f(k)+O(\log n)$. A similar fact holds for $(f(k)\cdot n^{O(1)})$-time-bounded DTMs that compute reductions; in this case we consider the problem of deciding whether a certain position in the output of the DTM contains a certain symbol. If the reduction is computed in time $f(k) \cdot n^{O(1)}$, then this problem can be decided by an ATM using space $f(k) + O(\log n)$. To finish the proof of the above claim, we (1) consider an ATM that model checks first-order formulas in space at most $f(k) \cdot \log n$ and time at most $f(k) \cdot n^{O(1)}$ with respect to the number of distinct variables as the parameter, and (2) modify it to run the above machine for the reduction whenever it wants to access an input symbol.

For the other direction, let (P, κ) be solvable by an ATM M using time $f(\kappa) \cdot n^c$ and space $f(\kappa) \cdot \log n$ for a computable function $f \colon \mathbb{N} \to \mathbb{N}$ and constant c on length-n inputs. In order to describe the reduction's construction we consider an input $x \in \Sigma^*$. We assume, without loss of generality, that M alternates between existential and universal states in each transition. That means, if we consider the start configuration as having time stamp 0, configurations with an even time stamp are always existential and configurations with an odd time stamp are always universal. Recall that a *configuration* C of M on input x consists of the current state, the head positions on the input tape and on the working tape, and the content of the working tape. Commonly a configuration is encoded as a (binary) string of length at most $c_M + f(k) \cdot \log n$ where c_M is a constant depending on M.

We present an fpt-reduction from (P, κ) to p-var-MC(FO). A first attempt for the reduction is to construct the (acyclic) *configuration graph* $G_M(x) = (\textsc{vert}, \textsc{edg})$ that contains all possible configurations of M as vertices and (directed) edges representing transitions between them. Moreover, the initial configuration is colored using a unary predicate I, and the existential and universal configurations are colored using unary predicates EXIST and UNIV, respectively. Then we state a formula φ_M that defines the acceptance behavior of ATMs, which is an alternating reachability query, that run in time at most $f(\kappa) \cdot n^c$ based on the graph. While we only need a constant number of variables for the formula, the graph considered in this reduction is too large to be constructible using an fpt-reduction since we consider all possible $2^{c_M + f(\kappa)\cdot \log n}$ configurations.

To get an fpt-reduction, we trade number of variables of the formula for the size of the constructed structure: Instead of constructing the graph explicitly, we modify the formula φ_M to a formula $\varphi_{M'}$ that not only defines the acceptance behavior of M, but also implicitly the configuration graph. How to modify the formula as well as how to construct a structure for this approach is described below.

Instead of constructing a configuration graph, the second version of our reduction produces a logical structure $\mathcal{A}$ with universe $U := \{1, \dots, f(\kappa) \cdot n^c\}$. The only relation on these elements is the bit predicate $\textsc{bit} = \{(i, j) \mid \text{position } i \text{ in bit-string } \operatorname{enc}(j) \text{ is } 1\}$.

The formula φ_M uses variables to store pointers to whole configurations. In order to avoid this, we encode configurations of M using substrings for a configuration's state, head position, and working tape content. To encode a configuration with the help of a formula's element variables, we replace each element variable x in φ_M by a group of variables consisting of a single variable x_{state} to contain the index of the state (assuming that the input size is large enough), a variable $x_{\text{in-head}}$ to encode the head position on the input tape, a variable $x_{\text{work-head}}$ to encode the head position on the work tape, and $f(\kappa)$ variables $x_{\text{content-}i}$ to encode the content of the ith length-$\log n$ block on the working tape. Finally, we replace the predicate symbols that are used to access the edges and vertex colorings of the graph by subformulas that define these predicates based on the predicate $\textsc{bit}$. The details of this well-known approach from descriptive complexity are described in the book of Immerman [13].

Both the formula φ'_M and the used structure can be constructed in time $f(k) \cdot n^{O(1)}$, and the number of variables used in φ'_M is bounded in terms of the machine M and the parameter of the input instance k. ◀

Based on translating the alternating Turing machines from Lemma 5 into circuits, it is possible to get a characterization of SXP in terms of families of Boolean circuits that are uniform (the building blocks of the circuits can be recognized, for example, by using a parameterized version of the classical class ALOGTIME where the parameter is given in an appropriate way to the uniformity machine). These circuits have size $n^{f(k)}$ and depth $f(k) \cdot n^{O(1)}$. Thus, their shape also supports our intuition that SXP is the right analog of PSPACE in the parameterized setting.

5 Parameterized polynomial-space-complete problems

Kasai, Adachi and Iwata [14] introduced a simple pebble game to provide a combinatorial characterization of different complexity classes by playing several variants of that game. An instance of the pebble game consists of a set of nodes X, a set of start positions $S \subseteq X$ for $k = |S|$ pebbles, a goal node $\gamma \in X$ and a set $R \subseteq X^3$ of rules which are triples of pairwise distinct nodes. There are two players in the game, which alternately move a pebble on the game board according to some rule $(u, v, w) \in R$: if there are pebbles on u and v but not on w, then the corresponding player can move the pebble from u to w. One player wins the game if he puts a pebble on the goal node or reaches a position where the other player is unable to move. The game board is *acyclic* if the underlying dag with vertex set X and arcs (u, w), (v, w) for all $(u, v, w) \in R$ is acyclic. In the acyclic pebble game the game board is required to be acyclic. It turns out that determining the winner in the pebble game is complete for EXPTIME and determining the winner in the acyclic variant is PSPACE-complete [14]. If one fixes the number of pebbles k, it is possible to determine the winner (in both variants) in time $n^{O(k)}$. This can easily be verified as the game can be simulated by an alternating Turing machine that uses $O(k \log n)$ space to store the current position of the pebbles. Adachi, Iwata and Kasai [2] proved a corresponding lower bound in the non-acyclic case. They simulated single-tape Turing machines of running time $O(n^k)$ within the $(2k+1)$-pebble game and used the time hierarchy theorem to obtain a lower bound of $n^{\Omega(k)}$. As remarked by Downey and Fellows [8] it follows that, parameterized by the number of pebbles, this problem is XP-complete. Thus, the pebble game supports the intuition that natural parameterizations of

EXPTIME-complete problems tend to be XP-complete. We show that the PSPACE-complete acyclic variant is complete for SXP under the same parameterization.

▶ **Theorem 6.** *Playing the pebble game on acyclic boards parameterized by the number of pebbles is complete for* SXP *under* fpt*-reductions.*

Proof. As the acyclic k-pebble game ends after a linear number of rounds, it can be simulated by an alternating Turing machine in space $O(k \log n)$ and time $O(n)$. Hence, this problem is in SXP. For the other direction we reduce from p-var-MC(FO). Let φ be a k-variable first-order formula and $\mathcal{A}$ be a structure with universe $[n]$. First, by allowing negation everywhere in the formula, we eliminate conjunction and universal quantification. We reduce the model checking problem to the acyclic $(k+2)$-pebble game such that $\mathcal{A} \models \varphi$ if, and only if, Player 1 wins the pebble game. We introduce a special node $\_$ to be used as middle vertex in the rules $(u, \_, w)$. At the beginning of the game there is a pebble on this node, which cannot be moved during the game. The acyclic game board resembles the structure of the formula. We use k pebbles to store the current assignment of the k variables and an additional pebble to control which subformula is evaluated. For every subformula ψ of φ we introduce a control node $X[\psi]$ and in addition nodes $X[\psi, x, v]$, for all variables $x \in \text{var}(\varphi)$ and elements $v \in [n]$, to store assignments of the variables. These nodes serve as the basic data structure on the game board. We define the rules and additional nodes by induction on the structure of the formula to satisfy the following invariant.

For every subformula $\psi(x_1, \dots, x_k)$, Player 1 wins from the pebble position $(X[\psi]$, $X[\psi, x_1, v_1]$, …, $X[\psi, x_k, v_k])$ iff $\mathcal{A} \models \psi[v_1, \dots, v_k]$.

As the pebble game is symmetric with respect to the players it follows that if the current pebble position is $(X[\psi]$, $X[\psi, x_1, v_1]$, …, $X[\psi, x_k, v_k])$ and it is Player 2's turn, then Player 1 wins iff $\mathcal{A} \not\models \psi[v_1, \dots, v_k]$. Initially, the k value pebbles are on the nodes $X[\varphi, x_1, v]$, …, $X[\varphi, x_k, v]$ for an arbitrary element v and the control pebble is on $X[\varphi]$. Thus, by the invariant above, Player 1 wins iff $\mathcal{A} \models \varphi$.

Atoms: For the base case let $\psi = R(x_{i_1}, \dots, x_{i_r})$ be an atom. We introduce additional nodes $Y[\psi, a_1, \dots, a_r]$ and $Y_p[\psi, x_{i_j}, a_1, \dots, a_r]$ for every tuple $(a_1, \dots, a_r) \in R^{\mathcal{A}}$, every variable x_{i_j}, $j \in [r]$ and $p \in \{1, 2\}$. There are rules $(X[\psi], \_, Y[\psi, a_1, \dots, a_r])$ that enable Player 1 to choose a tuple $(a_1, \dots, a_r)$ from the relation $R^{\mathcal{A}}$ that is consistent with the assignment specified by the pebbles on the nodes $X[\psi, x_i, v_i]$. To check this consistency both players are forced to use the following set of rules in the predefined order. First Player 2 moves the pebble from $Y[\psi, a_1, \dots, a_r]$ to $Y_1[\psi, x_{i_1}, a_1, \dots, a_r]$ using $(Y[\psi, a_1, \dots, a_r], \_, Y_1[\psi, x_{i_1}, a_1, \dots, a_r])$. Then it is Player 1's turn and both players alternately move the pebble using the rules $(Y_1[\psi, x_{i_j}, a_1, \dots, a_r]$, $X[\psi, x_{i_j}, a_j]$, $Y_2[\psi, x_{i_{j+1}}$, $a_1, \dots, a_r])$ and $(Y_2[\psi, x_{i_{j+1}}, a_1, \dots, a_r], \_, Y_1[\psi, x_{i_{j+1}}, a_1, \dots, a_r])$ for $j = 1, \dots, r-1$. Finally, there is a rule $(Y[\psi, x_{i_r}, a_1, \dots, a_r], \_, \gamma)$ that allows Player 1 to pebble the goal. By definition, this sequence of rules can be applied (and thus Player 1 wins the game as the goal node γ is pebbled) if $\mathcal{A} \models \psi[v_1, \dots, v_k]$. On the other hand, if $\mathcal{A} \not\models \psi[v_1, \dots, v_k]$, then Player 1 gets stuck and Player 2 wins.

Disjunction: If $\psi = \psi_1 \vee \psi_2$, then we have to ensure that from the pebble position $(X[\psi], X[\psi, x_1, v_1], \dots, X[\psi, x_k, v_k])$ Player 1 can move to either $(X[\psi_1], X[\psi_1, x_1, v_1], \dots,$ $X[\psi_1, x_k, v_k])$ or $(X[\psi_2], X[\psi_2, x_1, v_1], \dots, X[\psi_2, x_k, v_k])$. To make this decision, we introduce nodes $X_p[\psi_j]$, for $p, j \in \{1, 2\}$, and rules $(X[\psi], \_, X_1[\psi_j])$ and $(X_1[\psi_j], \_, X_2[\psi_j])$ for $j \in \{1, 2\}$. Thus, Player 1 can choose an j and move to $X_1[\psi_j]$. Afterwards, Player 2 is forced to move to $X_2[\psi_j]$. It remains to copy the current assignment to the subformula, that is, the players have to move the pebbles from $X[\psi, x, v]$ to $X[\psi_j, x, v]$. We use nodes $X[\psi_j, i]$ for $i \in [k+1]$ to control this process. The first rule is $(X_2[\psi_j], \_, X[\psi_j, 1])$. For all $v \in [n]$ and $i \leq k$ there are rules $(X[\psi, x_i, v], X[\psi_j, i], X[\psi_j, x_i, v])$ for copying the value of x_i and $(X[\psi_j, i], X[\psi_j, v, x_i], X[\psi_j, i+1])$ to move the control pebble. Both players must cooperate and use these rules successively to move the pebbles from $X[\psi, x, v]$ to $X[\psi_j, x, v]$. At the end of this process the pebble position is $(X[\psi_j, k+1], X[\psi_j, x_1, v_1]), \ldots, X[\psi_j, x_k, v_k])$ and it is Player 2's turn. He is forced to use the final rule $(X[\psi_j, k+1], \_, X[\psi_j])$. This finishes the description of the $\vee$-case.

Negation: Let $\psi = \neg\psi'$. We simulate negation by changing the role of both players. That is, from the configuration $(X[\psi], X[\psi, x_1, v_1], \ldots, X[\psi, x_k, v_k])$ where it is Player 1's turn we want to force both players to reach the configuration $(X[\psi'], X[\psi', x_1, v_1], \ldots, X[\psi', x_k, v_k])$ where it is Player 2's turn. Changing is role of the players is rather easy as we just have to introduce a dummy rule $(X[\psi], \_, X[\psi'])$ to force one move of the control pebble. Afterwards the players have to move the assignment pebbles from $X[\psi, x, v]$ to $X[\psi', x, v]$. This copy process can be done in the same deterministic way as described in the $\vee$-case.

Existential quantification: Let $\psi = \exists x_j \psi'$. To model the existential quantifier we have to ensure that from a pebble position $(X[\psi], X[\psi, x_1, v_1], \ldots, X[\psi, x_j, v_j]), \ldots, X[\psi, x_k, v_k])$ Player 1 can choose an element $w \in [n]$ and reach the new position $(X[\psi'], X[\psi', x_1, v_1], \ldots, X[\psi', x_j, w]), \ldots, X[\psi, x_k, v_k])$. Copying the values of x_i for $i \neq j$ (moving the pebbles from $X[\psi, x_i, v_i]$ to $X[\psi', x_i, v_i]$) can be done in the same way as in the previous cases. Hence, we end up with a configuration where it is Player 1's turn and the pebbles are on $X[\psi', x_i, v_i]$ $(i \neq j)$, $X[\psi, x_j, v_j]$, and the control pebble is on an additional node $X_0[\psi]$. To change the value of x_j we use the following construction. Let $X_1[\psi]$ and $X_2[\psi]$ be two additional nodes. There is a rule $(X_0[\psi], \_, X_1[\psi])$ and for every $v \in [n]$ we add the following three rules: $(X[\psi, x_j, v], X_1[\psi], X_2[\psi])$, $(X_1[\psi], X_2[\psi], X[\psi', x_j, v])$, $(X_2[\psi], X[\psi', x_j, v], X[\psi'])$. First, Player 1 moves the pebble from $X_0[\psi]$ to $X_1[\psi]$. Afterwards, Player 2 is forced to move the pebble from $X[\psi, x_j, v_j]$ to $X_2[\psi]$. Now Player 1 can choose some $w \in [n]$ and move the pebble from $X_1[\psi]$ to $X[\psi', x_j, w]$. The last move is done by Player 2, who is forced to move the pebble from $X_2[\psi]$ to $X[\psi']$. ◀

Another example that shifts the correspondence between EXPTIME and PSPACE in the classical world to XP and SXP in the parameterized setting is resolution. Resolution is a well-known and intensively studied proof system to detect the unsatisfiability of a given formula in conjunctive normal form. Starting with the clauses from the CNF formula one iteratively derives new clauses using only one simple rule: The resolution rule for a variable X takes two clauses $\gamma \cup \{X\}$, $\delta \cup \{\neg X\}$ and resolves $\gamma \cup \delta$. The given CNF formula is unsatisfiable if, and only if, the empty clause can be derived. The *width* of a refutation is the maximal number of literals in every clause of the derivation. A resolution derivation can naturally be viewed as a directed acyclic graph (dag) where the nodes are labeled with clauses and arcs pointing from the resolvent to its parents. The *depth* of a refutation is the length of the longest path in the corresponding dag. If on every path in this derivation dag no variable has been used twice by the resolution rule, then the derivation is *regular*. Note that the depth of every regular resolution refutation is at most linear in the number of variables, thus *linear depth resolution* generalizes regular resolution.

A resolution refutation of width k can be found be an alternating $O(k \log n)$-space Turing machine as follows: In each step, the machine stores one clause (using $k \log n$ bits) and tries to justify that this clause can be derived. Starting from the empty clause, the machine existentially guesses its parents and then universally chooses a parent to justify that it can be derived. The machine accepts the input, if the current clause is from the CNF formula

Table 1 To compare the correspondence between tractable and intractable in classical and parameterized complexity we denote by *poly* and *exp* polynomial and exponential growth, and by *fpt* and *xp* growth of the form $f(k) \cdot n^{O(1)}$ and $n^{f(k)}$, respectively.

Alternating Turing machines	EXPTIME	Alternating PSPACE machines of *exp* running time.
	PSPACE	Alternating PSPACE machines of *poly* running time.
	XP	Alternating SPACE$(f(k)\log n)$ machines of *xp* running time.
	SXP	Alternating SPACE$(f(k)\log n)$ machines of *fpt* running time.
Pebble games	EXPTIME	Pebble game.
	PSPACE	Acyclic pebble game.
	XP	Pebble game, parameter: the number of pebbles.
	SXP	Acyclic pebble game, parameter: the number of pebbles.
Resolution	EXPTIME	Bounded width resolution.
	PSPACE	Bounded width linear depth resolution.
	XP	Bounded width resolution, parameter: the width.
	SXP	Bounded width linear depth resolution, parameter: the width.

and rejects after $(2n)^k$ steps (which upper bounds the depth of width-k refutations). If we additionally require the depth to be linear, the alternating machine is able to find a refutation of width k in linear time. It follows that finding resolution refutations of width k is in EXPTIME, if k is part of the input, and in XP, if k is the parameter. Furthermore, finding linear depth refutations of width k is in PSPACE, if k is part of the input, and in SXP, if k is the parameter. By reducing the pebble game to bounded width resolution [3] it was shown that the corresponding problems are complete for EXPTIME, XP and PSPACE. We now show that the same reduction, stated in Fact 7, can be used to show that finding linear depth resolution refutations of bounded width is SXP-complete.

▶ **Fact 7** ([3]). *There is an* fpt*-reduction that takes an instance of the k-pebble game and produces a 3-*CNF *formula Γ such that Player 1 wins the k-pebble game iff Γ has a resolution refutation of width $k+1$. If in addition the game board is acyclic, then Γ has a regular resolution refutation of width $k+1$.*

▶ **Lemma 8.** *Finding linear depth resolution refutations of width k is complete for* SXP*.*

Proof. We already have observed that this problem is contained in SXP. Note that the reduction stated in Fact 7 reduces the parametrized acyclic pebble game to parameterized linear depth resolution, as every regular refutation has always linear depth. By Theorem 6 it follows that finding linear depth refutations is complete for SXP when parameterized by the width. ◀

6 Conclusion

We placed the model checking problem for fixed variable first-order logic within the hierarchy of intractable problems in parameterized complexity. As a consequence we exhibited a new parameterized complexity class, SXP, that corresponds to PSPACE in the same way as XP corresponds to EXPTIME. To support this intuition we gave characterizations in terms of alternating Turing machines, pebble games, and resolution refutations. The results are summarized in Table 1.

References

1 Karl A. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logic*, 73(3):235–276, 1995.

2 Akeo Adachi, Shigeki Iwata, and Takumi Kasai. Some combinatorial game problems require $\Omega(n^k)$ time. *J. ACM*, 31:361–376, March 1984.

3 Christoph Berkholz. On the complexity of finding narrow proofs. In *Foundations of Computer Science, IEEE Annual Symposium on*, pages 351–360, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

4 Christoph Berkholz. Lower bounds for existential pebble games and k-consistency tests. *Logical Methods in Computer Science*, 9(4), 2013.

5 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

6 Yijia Chen, Jörg Flum, and Martin Grohe. Machine-based methods in parameterized complexity theory. *Theoretical Computer Science*, 339(2–3):167–199, 2005.

7 R. Downey and M. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

9 Rodney G. Downey, Michael R. Fellows, and Ken Regan. Descriptive complexity and the W-hierachy. In *Proof Complexity and Feasible Arithmetic*, volume 39 of *AMS-DIMACS*, pages 119–134. AMS, 1998.

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

11 Jörg Flum and Martin Grohe. Model-checking problems as a basis for parameterized intractability. *Logical Methods in Computer Science*, 1(1), 2005.

12 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

13 Neil Immerman. *Descriptive complexity*. Springer, New York, 1999.

14 Takumi Kasai, Akeo Adachi, and Shigeki Iwata. Classes of pebble games and complete problems. *SIAM J. Comput.*, 8(4):574–586, 1979.

15 Richard E Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7:18–20, 1975.

16 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.

17 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

18 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC 1982)*, pages 137–146. ACM, 1982.