Available online at www.sciencedirect.com

SCIENCE DIRECT®

Annals of Pure and Applied Logic 130 (2004) 3–31

ANNALS OF PURE AND APPLIED LOGIC

www.elsevier.com/locate/apal

The complexity of first-order and monadic second-order logic revisited

Markus Frick[a], Martin Grohe[b,*]

[a]SAP AG, Neurottstr. 15a, 69190 Walldorf, Germany
[b]Institut für Informatik, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany

Available online 20 July 2004

Abstract

The model-checking problem for a logic L on a class C of structures asks whether a given L-sentence holds in a given structure in C. In this paper, we give super-exponential lower bounds for fixed-parameter tractable model-checking problems for first-order and monadic second-order logic.

We show that unless PTIME $=$ NP, the model-checking problem for monadic second-order logic on finite words is not solvable in time $f(k) \cdot p(n)$, for any *elementary* function f and any polynomial p. Here k denotes the size of the input sentence and n the size of the input word. We establish a number of similar lower bounds for the model-checking problem for first-order logic, for example, on the class of all trees.

1. Introduction

1.1. Model-checking problems

We study the complexity of a fundamental algorithmic problem, the so-called *model-checking* problem: given a sentence φ of some logic L and a structure $\mathcal{A}$, decide whether φ holds in $\mathcal{A}$. Model-checking and closely related problems are of importance in several areas of computer science, for example, in database theory, artificial intelligence, and automated verification. In this paper, we prove new lower bounds on the complexity of the model-checking problems for first-order and monadic second-order logic.

It is known that model-checking for both first-order and monadic second-order logic is PSPACE-complete [17,20] and thus most likely not solvable in polynomial time. While this

[*] Corresponding author.
E-mail addresses: markus.frick@sap.com (M. Frick), grohe@informatik.hu-berlin.de (M. Grohe).

0168-0072/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.apal.2004.01.007

result shows that the problems are intractable *in general*, it does not say too much about their complexity in practical situations. Typically, we have to check whether a relatively *small* sentence holds in a *large* structure. For example, when evaluating a database query, we usually have a small query and a large database. Similarly, when verifying that a finite state system satisfies some property, the specification of the property in a suitable logic will usually be small compared to the huge state space of the system. When analysing the complexity of the problem, we should take this imbalance between the size of the input sentence and the size of the input structure into account.

1.2. Parameterized complexity theory

Parameterized complexity theory (see [5]) is a relatively new branch of complexity theory that provides the framework for a refined complexity analysis of problems whose instances consist of different parts that typically have different sizes. In this framework, a *parameterized problem* is a problem whose instances consist of two parts of sizes n and k, respectively. k is called the *parameter*, and the assumption is that k is usually small, small enough that an algorithm that is exponential in k may still be feasible. A parameterized problem is called *fixed-parameter tractable* if it can be solved in time $f(k) \cdot p(n)$ for an arbitrary computable function f and some polynomial p. The motivation for this definition is that, since k is assumed to be small, the feasibility of an algorithm for the problem mainly depends on its behaviour in terms of n. Under this definition, a running time of $O(2^k \cdot n)$ is considered tractable, but running times of $O(n^k)$ or $O(k \cdot 2^n)$ are not, which seems reasonable.

Let us remark that although fixed-parameter tractability has proven to be a valuable concept allowing fine distinctions on the borderline between tractability and intractability, it seems somewhat questionable to admit *all* computable functions f for the parameter dependence of a fixed-parameter tractable algorithm. If f is doubly exponential or worse, an $O(f(k) \cdot n)$-algorithm can hardly be considered tractable. The main contribution of this paper to parameterized complexity theory is to show that there are natural fixed-parameter tractable problems requiring parameter dependences f that are doubly exponential or even non-elementary.

1.3. The parameterized complexity of model-checking problems

Model-checking problems have a natural parameterization in which the size k of the input sentence is the parameter. We have argued above that k is usually small in the practical situations we are interested in, so a parameterized complexity analysis is appropriate. Unfortunately, it turns out that the model-checking problem for first-order logic is complete for the parameterized complexity class AW[$*$], which is conjectured to strictly contain the class FPT of all fixed-parameter tractable problems. Thus probably model-checking for first-order logic is not fixed-parameter tractable. Of course this implies that model-checking for the stronger monadic second-order logic is also most-likely not fixed-parameter tractable. As a matter of fact, it follows immediately from the observation that there is a monadic second-order sentence saying that a graph is 3-colourable that model-checking for monadic second-order logic is not fixed-parameter tractable unless P $=$ NP.

It is interesting to compare these intractability results for first-order logic and monadic second-order logic with the following: the model-checking problem for linear time temporal logic LTL is solvable in time $2^{O(k)} \cdot n$ [14], making it fixed-parameter tractable and also tractable in practise. On the other hand, model-checking for LTL is PSPACE-complete (as it is for first-order and monadic second-order logic). So parameterized complexity theory helps us in establishing an important distinction between problems of the same classical complexity.[1] We may argue, however, that the comparison between LTL model-checking and first-order model-checking underlying these results is slightly unfair. As the name linear time temporal logic indicates, LTL only speaks about a linearly ordered sequence of events. On an arbitrary structure, an LTL formula can thus only speak about the paths through the structure. First-order formulas do not have such a restricted view. It is therefore more interesting to compare LTL and first-order logic on *words*, which are the natural structures describing linear sequences of events. A well-known result of Kamp [12] states that LTL and first-order logic have the same expressive power on words. And indeed, model-checking for first-order logic and even for monadic second-order logic is fixed-parameter tractable if the input structures are restricted to be words. This is a consequence of Büchi's theorem [2], saying that for every sentence of monadic second-order logic one can effectively find a finite automaton accepting exactly those words in which the sentence holds. A fixed-parameter tractable algorithm for monadic second-order model-checking on words may proceed as follows: it first translates the input sentence into an equivalent automaton and then tests in linear time whether this automaton accepts the input word. But note that since there is no elementary bound for the size of a finite automaton equivalent to a given first-order or monadic second-order sentence [18] (also see [15]), the parameter dependence of this algorithm is non-elementary, thus it does not even come close to the $2^{O(k)} \cdot n$ model-checking algorithm for LTL. Of course this does not rule out the existence of other, better fixed-parameter tractable algorithms for first-order or monadic second-order model-checking.

1.4. Our results

Our first theorem shows that there is no fundamentally better fixed-parameter tractable algorithm for first-order and monadic second-order model-checking on the class of words than the automata based one described in the previous paragraph.

Theorem 1. (1) *Assume that* PTIME $\neq$ NP. *Let f be an elementary function and p a polynomial. Then there is no model-checking algorithm for monadic second-order logic on the class of words whose running time is bounded by $f(k) \cdot p(n)$.*

(2) *Assume that* FPT $\neq$ AW[$*$]. *Let f be an elementary function and p a polynomial. Then there is no model-checking algorithm for first-order logic on the class of words whose running time is bounded by $f(k) \cdot p(n)$.*

[1] A critical reader may remark that this distinction between the complexities of LTL model-checking and first-order model-checking was known before anybody thought of parameterized complexity-theory. This is true, but how can we be sure that there is no $2^{O(k)} \cdot n$ model-checking algorithm for first-order model-checking? The role of parameterized-complexity theory is to give evidence for this.

Here k denotes the size of the input sentence of the model-checking problem and n the size of the input word.

Recall that a function $f : \mathbb{N} \to \mathbb{N}$ is *elementary* if it can be formed from the successor function, addition, subtraction, and multiplication using compositions, projections, bounded additions and bounded multiplications (of the form $\sum_{z \leq y} g(\bar{x}, z)$ and $\prod_{z \leq y} g(\bar{x}, z)$). The crucial fact for us is that a function f is bounded by an elementary function if, and only if, it is bounded by an h-fold exponential function for some fixed h (see, for example, [4]).

To prove the theorem, we use similar coding tricks as those that can be used to prove that there is no elementary algorithm for deciding the satisfiability of first-order sentences over words [18].

Model-checking for first-order and monadic second-order logic is known to be fixed-parameter tractable on several other classes of structures besides words: model-checking for monadic second-order logic is also fixed-parameter tractable on trees and graphs of bounded tree-width [3]. The latter is a well-known theorem due to Courcelle [3] playing a prominent role in parameterized complexity theory. Theorem 1 implies that the parameter dependence of monadic-second-order model-checking on trees and and graphs of bounded tree-width is also non-elementary. In addition to trees and graphs of bounded tree-width, model-checking for first-order logic is fixed-parameter tractable on further interesting classes of graphs such as graphs of bounded degree [16], planar graphs [10], and more generally locally tree-decomposable classes of structures [10]. Theorem 1(2) does *not* imply lower bounds for the parameter dependence here. The reason for that is a peculiar detail in the encoding of words by relational structures. The standard encoding includes the linear order of the letters in a word as an explicit relation of the structure. If we omit the order and just include a successor relation, Theorem 1(1) still holds, because the order is definable in monadic second-order logic. However, the order is not definable in first-order logic, and Theorem 1(2) does not extend to words without order. Indeed, we give a model-checking algorithm for first-order logic on words without order, and more generally on structures of degree 2, with a running time $2^{2^{O(k)}} \cdot n$, that is, with a doubly exponential parameter dependence. We also give a model-checking algorithm for first-order logic on structures of bounded degree $d \geq 3$ with a triply exponential parameter dependence. We match these upper bounds by corresponding lower bounds:

Theorem 2. *Assume that* FPT $\neq$ AW[$*$], *and let p be a polynomial.*

(1) *There is no model-checking algorithm for first-order logic on the class of words without order whose running time is bounded by*

$$2^{2^{o(k)}} \cdot p(n).$$

(2) *There is no model-checking algorithm for first-order logic on the class of binary trees whose running time is bounded by*

$$2^{2^{2^{o(k)}}} \cdot p(n).$$

Again, k denotes the size of the input sentence and n the size of the input structure.

Finally, we obtain a non-elementary lower bound for first-order model-checking on trees, which implies lower bounds for planar graphs and all other classes of graphs that contain all trees.

Theorem 3. *Assume that* FPT $\neq$ AW[$*$]. *Let f be an elementary function and p a polynomial. Then there is no model-checking algorithm for first-order logic on the class of trees whose running time is bounded by $f(k) \cdot p(n)$.*

2. Preliminaries

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. *All structures considered in this paper are assumed to have a finite universe*. The *reduct* of a τ-structure $\mathcal{A}$ to a vocabulary $\tau' \subseteq \tau$ is the τ'-structure with the same universe as $\mathcal{A}$ and the same interpretation of all symbols in τ'. An expansion of a structure $\mathcal{A}$ is a structure $\mathcal{A}'$ such that $\mathcal{A}$ is a reduct of $\mathcal{A}'$. In particular, if $\mathcal{A}$ is a structure and $a \in A$, then by $(\mathcal{A}, a)$ we denote the expansion of $\mathcal{A}$ by the constant a. We write $\mathcal{A} \cong \mathcal{B}$ to denote that structures $\mathcal{A}$ and $\mathcal{B}$ are isomorphic.

Let Σ be a finite alphabet. We let $\tau(\Sigma)$ be the vocabulary consisting of a binary relation symbol $\leq$, a unary function symbol S, two constant symbols 'min' and 'max', and a unary relation symbol P_s for every $s \in \Sigma$. A *word structure* over Σ is a $\tau(\Sigma)$-structure $\mathcal{W}$ with the following properties:

- $\leq^{\mathcal{W}}$ is a linear order of W, $\min^{\mathcal{W}}$ and $\max^{\mathcal{W}}$ are the minimum and maximum element of $\leq^{\mathcal{W}}$, and $S^{\mathcal{W}}$ is the successor function associated with $\leq^{\mathcal{W}}$, where we let $S^{\mathcal{W}}(\max^{\mathcal{W}}) = \max^{\mathcal{W}}$.
- For every $a \in W$ there exists precisely one $s \in \Sigma$ such that $a \in P_s^{\mathcal{W}}$.

We refer to elements $a \in W$ as the *positions* in the word (structure) and, for every position $a \in W$, to the unique s such that $a \in P_s^{\mathcal{W}}$ as the *letter at a*.

It is obvious how to associate a word from the set Σ^* of all words over Σ with every word structure over Σ and, conversely, how to associate an up to isomorphism unique word structure with every word in Σ^*. We identify words with the corresponding word structures and write $\mathcal{W} \in \Sigma^*$ to refer both to the word and the structure.

The class of all words (or word structures) over any alphabet is denoted by $\mathbb{W}$. The length of a word $\mathcal{W}$ is denoted by $|\mathcal{W}|$.

A *subword* of a word $\mathcal{W} = s_0 \dots s_{n-1} \in \Sigma^*$ is either the empty word or a word $s_i \dots s_j$ for some $i, j, 0 \leq i \leq j < n$. We write $\mathcal{V} \sqsubseteq \mathcal{W}$ to denote that $\mathcal{V}$ is a subword of $\mathcal{W}$.

We assume that the reader is familiar with propositional logic, first-order logic FO and monadic second-order logic MSO (see, for example, [7]). If θ is a formula of propositional logic and α is a truth-value assignment to the variables of θ, then we write $\alpha \models \theta$ to denote that α satisfies θ. Similarly, if $\varphi(x_1, \dots, x_k)$ is a first-order or monadic second-order formula with free variables $x_1, \dots, x_k$, $\mathcal{A}$ is a structure, and $a_1, \dots, a_k \in A$, then we write $\mathcal{A} \models \varphi(a_1, \dots, a_k)$ to denote that $\mathcal{A}$ satisfies φ if the variables $x_1, \dots, x_k$ are interpreted by $a_1, \dots, a_k$, respectively. A *sentence* is a formula without free variables. The *quantifier-rank* of a formula φ, that is, the maximum number of nested quantifiers in φ, is denoted by $\mathrm{qr}(\varphi)$.

The *model-checking problem* for a logic L on a class C of structures, denoted by $\mathrm{MC}(\mathrm{L}, \mathrm{C})$, is the following decision problem:

Input:	Structure $\mathcal{A} \in \mathrm{C}$, sentence $\varphi \in \mathrm{L}$
Problem:	Decide if $\mathcal{A} \models \varphi$.

We fix a reasonable encoding of structures and formulas by words over $\{0, 1\}$. We denote the length of the encoding of a structure $\mathcal{A}$ by $\|\mathcal{A}\|$ and the length of the encoding of a formula φ by $\|\varphi\|$. When reasoning about model-checking problems, we usually use n to denote the size $\|\mathcal{A}\|$ of the input structure and k to denote the size $\|\varphi\|$ of the input sentence.

Our underlying model of computation is the standard RAM-model with addition and subtraction as arithmetic operations (cf. [1,19]). In our complexity analysis we use the uniform cost measure.

It is well-known that if we are interested in the complexity of first-order or monadic second-order model-checking on words, the alphabet is inessential. This can be phrased as follows:

Fact 4. Let $\mathrm{L} \in \{\mathrm{FO}, \mathrm{MSO}\}$. Then there is a linear time algorithm that, given a sentence $\varphi \in \mathrm{L}$ and a word $\mathcal{W} \in \mathbb{W}$, computes a sentence $\varphi' \in \mathrm{L}$ of vocabulary $\tau(\{0, 1\})$ and a word $\mathcal{W}' \in \{0, 1\}^*$ such that $\|\varphi'\| \in O(\|\varphi\|)$, $\|\mathcal{W}'\| \in O(\|\mathcal{W}\|)$, and $(\mathcal{W} \models \varphi \iff \mathcal{W}' \models \varphi')$.

$\mathbb{N}$ denotes the set of natural numbers (including 0). For all $n, i \in \mathbb{N}$ we let $\mathrm{bit}(i, n)$ denote the ith bit in the binary representation of n. (Here we count the lowest priority bit as the 0th bit.) lg denotes the base-2 logarithm, and, for $i \in \mathbb{N}$, $\lg^{(i)}$ denotes the i-fold logarithm. More formally, $\lg^{(i)}$ is defined by $\lg^{(0)}(n) = n$ and $\lg^{(i+1)}(n) = \lg \lg^{(i)}(n)$.

We define the *tower function* $T : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ by $T(0, r) = r$ and $T(h+1, r) = 2^{T(h,r)}$ for all $h \in \mathbb{N}$, $r \in \mathbb{R}$. Thus $T(h, r)$ is a tower of 2s of height h with an r sitting on top. Observe that for all $n, h \in \mathbb{N}$ with $n \geq 1$ we have $T(h, \lg^{(h)} n) = n$.

3. Succinct encodings

We introduce a sequence of encodings μ_h, for $h \geq 1$, of natural numbers by words over certain finite alphabets. They are more and more "succinct" not in the sense that the codewords are shorter and shorter, but in the sense that they can be "decoded" by shorter and shorter first-order formulas. Decoding is actually said too much here, what we mean is that there are shorter and shorter first-order formulas stating that two words encode the same number. For example, if we encode numbers in unary, for every n there is a first-order formula of length $O(n)$ stating that two words encode the same number smaller than 2^n. If we encode numbers in binary, there is a first-order formula of length $O(n)$ stating that two words encode the same number smaller than 2^{2^n}. We shall give, for every $h \geq 0$, an encoding such that for every n there is a first-order formula of length $O(n)$ stating that two words encode the same number smaller than $T(h,n)$. This is what Lemma 8, the key result of this section, states.

For all $h \geq 1$ we let $\Sigma_h = \{\texttt{0}, \texttt{1}, \texttt{<1>}, \texttt{</1>}, \ldots, \texttt{<h>}, \texttt{</h>}\}$. The “tags” `<i>` and `</i>` represent single letters of the alphabet and are just chosen to improve readability. We define $L : \mathbb{N} \to \mathbb{N}$ by $L(0) = 0$, $L(1) = 1$, $L(n) = \lfloor \lg(n-1) \rfloor + 1$ for $n \geq 2$. Note that for $n \geq 1$, $L(n)$ is precisely the length of the binary representation of $n - 1$.

We are now ready to define our encodings $\mu_h : \mathbb{N} \to \Sigma_h^*$, for $h \geq 1$. We let $\mu_1(0) = \texttt{<1></1>}$ and

$$\mu_1(n) = \texttt{<1>}\,\mathrm{bit}(0, n-1)\,\mathrm{bit}(1, n-1) \ldots \mathrm{bit}(L(n)-1, n-1)\,\texttt{</1>}$$

for $n \geq 1$. For $h \geq 2$, we let $\mu_h(0) = \texttt{<h></h>}$ and

$$\begin{array}{rll} \mu_h(n) = & \texttt{<h>} & \\ & & \mu_{h-1}(0)\ \mathrm{bit}(0, n-1) \\ & & \mu_{h-1}(1)\ \mathrm{bit}(1, n-1) \\ & & \quad\vdots \\ & & \mu_{h-1}(L(n)-1)\ \mathrm{bit}(L(n)-1, n-1) \\ & \texttt{</h>} & \end{array}$$

for $n \geq 1$. Here empty spaces and line breaks are just used to improve readability.

Example 5.

$$\begin{array}{rlllll} \mu_2(47) = & \texttt{<2>} & & = & \texttt{<2>} & \\ & & \mu_1(0)\ \texttt{0} & & & \texttt{<1></1>}\ \texttt{0} \\ & & \mu_1(1)\ \texttt{1} & & & \texttt{<1>0</1>}\ \texttt{1} \\ & & \mu_1(2)\ \texttt{1} & & & \texttt{<1>1</1>}\ \texttt{1} \\ & & \mu_1(3)\ \texttt{1} & & & \texttt{<1>01</1>}\ \texttt{1} \\ & & \mu_1(4)\ \texttt{0} & & & \texttt{<1>11</1>}\ \texttt{0} \\ & & \mu_1(5)\ \texttt{1} & & & \texttt{<1>001</1>}\ \ \texttt{1} \\ & \texttt{</2>} & & & \texttt{</2>} & \end{array}$$

Lemma 6.

$$|\mu_h(n)| \in O(h \cdot \lg^2 n).$$

Proof. We define functions $L_i : \mathbb{N} \to \mathbb{N}$ as follows: $L_1(n) = L(n)$ for all $n \in \mathbb{N}$ and $L_i(n) = L_{i-1}(L(n))$ for all $i, n \in \mathbb{N}$ with $i \geq 2$. Moreover, we define $P_i : \mathbb{N} \to \mathbb{N}$ for $i \geq 1$ by

$$P_i(n) = \prod_{j=1}^{i} L_j(n).$$

Observe that for all $i \geq 2$ and $n \geq 1$ we have $P_i(n) = L(n) \cdot P_{i-1}(L(n))$.

We first prove, by induction on $h \geq 1$, that for all $n \geq 1$,

$$|\mu_h(n)| \leq 4h \cdot P_h(n). \tag{1}$$

We have $\mu_1(n) = 2 + L(n) \le 4L(n) = 4P_1(n)$, so (1) is true for $h = 1$. Let $h \ge 2$ an suppose that (1) holds for $h - 1$. Then

$$\begin{aligned}
|\mu_h(n)| &= 2 + L(n) + \sum_{i=0}^{L(n)-1} |\mu_{h-1}(i)| \\
&= 2 + L(n) + 2 + \sum_{i=1}^{L(n)-1} |\mu_{h-1}(i)| \\
&\le 4 + L(n) + \sum_{i=1}^{L(n)-1} 4(h-1) \cdot P_{h-1}(i) \\
&\le 4 + L(n) + 4(L(n)-1) \cdot (h-1) \cdot P_{h-1}(L(n)) \\
&\le L(n) + 4(h-1) \cdot L(n) \cdot P_{h-1}(L(n)) \\
&\le L(n) + 4(h-1) \cdot P_h(n) \\
&\le 4h \cdot P_h(n).
\end{aligned}$$

This proves (1).

Since $L(n) \in \Theta(\lg n)$, to complete the proof of the lemma it suffices to show that there is a constant c such that for all $h, n \ge 1$ we have $P_h(n) \le c \cdot L(n)^2$. Since $L(L(n)) \in O(\lg \lg n)$ and $L(n) \in \Omega(\lg n)$, there is an n_0 such that for all $n \ge n_0$ we have

$$L(L(n))^2 \le L(n).$$

Note that $P = \{P_h(m) \mid m < n_0, h \ge 1\}$ is a finite set and let $c = \max(P)$.

We prove that $P_h(n) \le c \cdot L(n)^2$ by induction on $h \ge 1$. Since $P_1(n) = L(n)$, this statement is true for $h = 1$. For $h \ge 2$, we have $P_h(n) = L(n) \cdot P_{h-1}(L(n))$. If $L(n) < n_0$, we have $P_{h-1}(L(n)) \le c$ and thus $P_h(n) \le cL(n)$. If $L(n) \ge n_0$, we have $L(L(n))^2 \le L(n)$. By induction hypothesis, $P_{h-1}(L(n)) \le c \cdot L(L(n))^2$. Thus

$$P_h(n) = L(n) \cdot P_{h-1}(L(n)) \le L(n) \cdot c \cdot L(L(n))^2 \le c \cdot L(n)^2. \quad \Box$$

Lemma 7. *There is an algorithm that, given $h, n \in \mathbb{N}$, computes $\mu_h(n)$ in time $O(|\mu_h(n)|) = O(h \cdot \lg^2 n)$.*

Proof. The algorithm computes $\mu_h(n)$ in a straightforward recursive manner. We get the following recurrence for the running time $R(h, n)$:

$$R(h, n) \le O(L(n)) + \sum_{i=0}^{L(n)} R(h-1, L(i)).$$

This recurrence is very similar to the one we obtained in the proof of Lemma 6 and can easily be solved using the same methods. □

Observe that for all $m \ge 1$ we have

$$2^m = \max\{n \in \mathbb{N} \mid L(n) \le m\}.$$

Recall that $T(h, \ell)$ is a tower of 2s of height h with an ℓ on top. Thus, in particular, for all $h, \ell \geq 1$ we have

$$T(h, \ell) = \max\{n \in \mathbb{N} \mid L(n) \leq T(h-1, \ell)\}. \tag{2}$$

Lemma 8. *Let $h \geq 1$, $\ell \geq 0$. There is a first-order formula $\chi_{h,\ell}(x, y)$ of size $O(h \cdot \lg h + \ell)$ such that for all words $\mathcal{W}$, $a, b \in W$, and $m, n \in \{0, \ldots, T(h, \ell)\}$ the following holds:*

If a is the first position of a subword $\mathcal{U} \sqsubseteq \mathcal{W}$ with $\mathcal{U} \cong \mu_h(m)$ and b is the first position of a subword $\mathcal{V} \sqsubseteq \mathcal{W}$ with $\mathcal{V} \cong \mu_h(n)$, then

$$\mathcal{W} \models \chi_{h,\ell}(a, b) \Longleftrightarrow m = n.$$

Furthermore, the formula $\chi_{h,\ell}$ can be computed from h and ℓ in time $O(h \cdot \lg h + \ell)$.

Proof. Let $h = 1$. Recall that the μ_1-encoding of an integer $p \geq 1$ is just the binary encoding of $p - 1$ enclosed in `<1>`, `</1>`. Hence to say that x and y are μ_1-encodings of the same numbers, we have to say that for all pairs $x + i, y + i$ of corresponding positions between x respectively y and the next closing `</1>`, there are the same letters at $x + i$ and $y + i$. For numbers p in $\{0, \ldots, T(1, \ell)\}$, there are at most $L(p) \leq \ell$ positions to be investigated. To express this, we let

$$\begin{aligned}\chi_{1,\ell}(x, y) = \exists x_1 \ldots \exists x_\ell \exists y_1 \ldots \exists y_\ell \\ \Bigg(Sx = x_1 \wedge \bigwedge_{i=1}^{\ell-1} ((P_{\texttt{</1>}} x_i \wedge x_i = x_{i+1}) \vee (\neg P_{\texttt{</1>}} x_i \wedge Sx_i = x_{i+1})) \\ \wedge\, Sy = y_1 \wedge \bigwedge_{i=1}^{\ell-1} ((P_{\texttt{</1>}} y_i \wedge y_i = y_{i+1}) \vee (\neg P_{\texttt{</1>}} y_i \wedge Sy_i = y_{i+1})) \\ \wedge \bigwedge_{i=1}^{\ell} ((P_0 x_i \leftrightarrow P_0 y_i) \wedge (P_1 x_i \leftrightarrow P_1 y_i)) \Bigg).\end{aligned}$$

Now let $h \geq 2$ and suppose that we have already defined $\chi_{h-1,\ell}(x, y)$. It will be convenient to have the following auxiliary formulas available:

$$\begin{aligned}\chi^h_{\text{int}}(x, y) &= x < y \wedge \forall z\, ((x < z \wedge z \leq y) \to \neg P_{\texttt{</h>}} z)\,, \\ \chi^h_{\text{last}}(x, y) &= x < y \wedge P_{\texttt{</h>}} y \wedge \forall z\, ((x < z \wedge z < y) \to \neg P_{\texttt{</h>}} z)\,.\end{aligned}$$

Intuitively, $\chi^h_{\text{int}}(x, y)$ says that y is in the interior of the subword of the form $\mu_h(p)$ starting at x and $\chi^h_{\text{last}}(x, y)$ says that y is the last position of the subword of the form $\mu_h(p)$ starting at x, provided such a subword indeed starts at x.

To say that the subwords starting at x and y are μ_h-encodings of the same numbers, we have to say that for all positions w between x and the next closing `</h>` and all positions z between y and the next closing `</h>`, if w and z are first positions of subwords isomorphic to $\mu_{h-1}(q)$ for some $q \in \mathbb{N}$, then the positions following these two subwords are either both `1`s or both `0`s. For all subwords of $\mu_h(p)$ of the form $\mu_{h-1}(q)$ we have $q \in \{0, \ldots, L(p)\}$. In order to apply the formula $\chi_{h-1,\ell}$ to test equality of such subwords, we must have $q \leq L(p) \leq T(h-1, \ell)$. By (2), the last inequality holds for all $p \leq T(h, \ell)$. Thus for such p we can use the formula $\chi_{h-1,\ell}$ to test equality of subwords of $\mu_h(p)$ of the form

$\mu_{h-1}(q)$. As a first approximation to our formula $\chi_{h,\ell}$, we let

$$\begin{aligned}
\chi'_{h,\ell}(x,y) = \forall w\Big(&\big(\chi^h_{\text{int}}(x,w) \wedge P_{\texttt{<h - 1>}}w\big)\\
&\qquad \to \exists z\big(\chi^h_{\text{int}}(y,z) \wedge P_{\texttt{<h - 1>}}z \wedge \chi_{h-1,\ell}(w,z)\big)\Big)\\
\wedge\ \forall z\Big(&\big(\chi^h_{\text{int}}(y,z) \wedge P_{\texttt{<h - 1>}}z\big)\\
&\qquad \to \exists w\big(\chi^h_{\text{int}}(x,w) \wedge P_{\texttt{<h - 1>}}w \wedge \chi_{h-1,\ell}(w,z)\big)\Big)\\
\wedge\ \forall w\forall z\Big(&\big(\chi^h_{\text{int}}(x,w) \wedge P_{\texttt{<h - 1>}}w \wedge \chi^h_{\text{int}}(y,z) \wedge P_{\texttt{<h - 1>}}z \wedge \chi_{h-1,\ell}(w,z)\big)\\
&\qquad \to \exists w'\exists z'\big(\chi^{h-1}_{\text{last}}(w,w') \wedge \chi^{h-1}_{\text{last}}(z,z') \wedge (P_1Sz' \leftrightarrow P_1Sw')\big)\Big).
\end{aligned}$$

The first line of this formula says that every subword of the form $\mu_{h-1}(q)$ in the subword of the form $\mu_h(p)$ starting at x also occurs in the subword of the form $\mu_h(p)$ starting at y. The second line says that every subword of the form $\mu_{h-1}(q)$ in the subword of the form $\mu_h(p)$ starting at y also occurs in the subword of the form $\mu_h(p)$ starting at x. The third and fourth lines say that if w and z are the first positions of isomorphic subwords of the form $\mu_{h-1}(q)$, then they are either both followed by a $\texttt{1}$ or both by a $\texttt{0}$ (since the only two letters that can appear immediately after a subword $\mu_{h-1}(q)$ in a word $\mu_h(p)$ are $\texttt{0}$ and $\texttt{1}$).

This formula says what we want, but unfortunately it is too large to achieve the desired bounds. The problem is that there are three occurrences of the subformula $\chi_{h-1,\ell}(w,z)$. We we can easily overcome this problem. We let

$$\zeta(w,z) = \exists w'\exists z'\left(\chi^{h-1}_{\text{last}}(w,w') \wedge \chi^{h-1}_{\text{last}}(z,z') \wedge P_1Sz' \leftrightarrow P_1Sw'\right)$$

and

$$\begin{aligned}
\chi_{h,\ell}(x,y) = \forall w\exists z\Bigg(&\big(\chi^h_{\text{int}}(x,w) \to \chi^h_{\text{int}}(y,z)\big)\\
&\wedge \big(\chi^h_{\text{int}}(y,w) \to \chi^h_{\text{int}}(x,z)\big)\\
&\wedge \big(P_{\texttt{<h - 1>}}w \to P_{\texttt{<h - 1>}}z\big)\\
&\wedge \Bigg(\Big(\big(\chi^h_{\text{int}}(y,w) \vee \chi^h_{\text{int}}(x,w)\big) \wedge P_{\texttt{<h - 1>}}w\Big)\\
&\qquad \to \chi_{h-1,\ell}(w,z) \wedge \zeta(w,z)\Bigg)\Bigg).
\end{aligned}$$

It is not hard to see that $\chi_{h,\ell}(x,y)$ has the desired meaning.

Observing that $\|\chi_{1,\ell}\| \in O(\ell)$ and that $\|\chi_{h,\ell}\| = \|\chi_{h-1,\ell}\| + c \cdot \lg h$ for some constant c, we obtain the desired bound on the size of the formulas. To see why we need the factor $\lg h$ here, recall that $\|\varphi_{h,\ell}\|$ is the length of a *binary* encoding of $\varphi_{h,\ell}$. The vocabulary of the formula $\varphi_{h,\ell}$ is of size $O(h)$, thus the binary encoding of the symbols in this vocabulary will require $O(\lg h)$ bits.

The fact that $\chi_{h,\ell}$ can be computed in time linear in the size of the output is immediate from the construction. □

4. Encodings of propositional formulas

In this section, we give a sequence encoding of propositional formulas in conjunctive normal form and assignments to the variables of these formulas such that there are shorter and shorter first-order formulas stating that the encoded assignment satisfies the encoded formula. The key idea is to use the encodings μ_h of the natural numbers to encode propositional variables by their index. Then by Lemma 8, we can check with a very short first-order formula if two subwords of a codeword that represent variables actually represent the same variable. This way we can look up the value of a variable in a table representing the assignment.

The class of all formulas in conjunctive normal form is denoted by CNF, and for every $k \geq 1$ the class of all formulas in k-conjunctive normal form, that is, conjunctions of clauses of size at most k, is denoted by k-CNF.

We assume that propositional formulas only contain variables X_i, for $i \in \mathbb{N}$. For a set Θ of propositional formulas, we let $\Theta(n)$ denote the set of all formulas in Θ whose variables are among $X_0, \dots, X_{n-1}$.

To encode formulas and assignments, we will use an alphabet that is obtained from the alphabet Σ_h introduced in the previous section by adding a number of symbols in several stages throughout this section. We start by adding the symbols

$$\texttt{+}, \texttt{-}, \texttt{<lit>}, \texttt{</lit>}, \texttt{<clause>}, \texttt{</clause>}, \texttt{<cnf>}, \texttt{</cnf>}.$$

We fix h and define an encoding of CNF-formulas by words as follows: For a literal λ, we let

$$\mu_h(\lambda) = \begin{cases} \texttt{<lit>}\ \mu_h(i)\ \texttt{+}\ \texttt{</lit>} & \text{if } \lambda = X_i \\ \texttt{<lit>}\ \mu_h(i)\ \texttt{-}\ \texttt{</lit>} & \text{if } \lambda = \neg X_i \end{cases}$$

(for every $i \in \mathbb{N}$). For a clause $\delta = (\lambda_1 \vee \cdots \vee \lambda_m)$ we let

$$\mu_h(\delta) = \texttt{<clause>}\ \mu_h(\lambda_1) \cdots \mu_h(\lambda_m)\ \texttt{</clause>},$$

and for a CNF-formula $\gamma = (\delta_1 \wedge \cdots \wedge \delta_m)$ we let

$$\mu_h(\gamma) = \texttt{<cnf>}\ \mu_h(\delta_1)\ \cdots\ \mu_h(\delta_m)\ \texttt{</cnf>}.$$

Next, we need to encode assignments. Let $A(n)$ denote the set of all assignments

$$\alpha : \{X_0, \dots, X_{n-1}\} \to \{\text{TRUE}, \text{FALSE}\}.$$

We add the symbols `<val>`, `</val>`, `<asn>`, `</asn>`, `true`, `false` to our alphabet. For an assignment $\alpha \in A(n)$, we let

$$\begin{aligned} \mu_h(\alpha) \quad = \texttt{<asn>} & \\ & \texttt{<val>}\mu_h(0)\,\alpha(X_0)\texttt{</val>} \\ & \quad\vdots \\ & \texttt{<val>}\mu_h(n-1)\,\alpha(X_{n-1})\texttt{</val>} \\ \texttt{</asn>}. & \end{aligned}$$

Of course what is meant by $\alpha(X_i)$ here is the symbol `true` if $\alpha(X_i) = \mathrm{TRUE}$ and the symbol `false` otherwise. For a pair $(\gamma, \alpha) \in \mathrm{CNF}(n) \times A(n)$ we simply let $\mu_h(\gamma, \alpha) = \mu_h(\gamma)\, \mu_h(\alpha)$.

The following lemma is an immediate consequence of Lemmas 6 and 7:

Lemma 9. *Let $h \in \mathbb{N}$ and $(\gamma, \alpha) \in \mathrm{CNF}(n) \times A(n)$. Then $|\mu_h(\gamma, \alpha)| = O(h \cdot \lg^2 n \cdot (\|\gamma\| + n))$ and there is an algorithm that computes $\mu_h(\gamma, \alpha)$ in time $O(h \cdot \lg^2 n \cdot (\|\gamma\| + n))$ (that is, linear in the size of the output).*

Lemma 10. *For all $h, \ell \in \mathbb{N}$ there is a first-order sentence $\varphi_{h,\ell}$ of size $O(h \cdot \lg h + \ell)$ such that for all $n \le T(h, \ell)$ and $(\gamma, \alpha) \in \mathrm{CNF}(n) \times A(n)$,*

$$\mu_h(\gamma, \alpha) \models \varphi_{h,\ell} \Longleftrightarrow \alpha \models \gamma.$$

Furthermore, the formula $\varphi_{h,\ell}$ can be computed in time $O(h \cdot \lg h + \ell)$.

Proof. Let $\chi_{h,\ell}(x, y)$ be the formula defined in Lemma 8. Recall that it says that the subwords of the form $\mu_h(m)$ and $\mu_h(n)$ starting at x, y, respectively, are identical, provided that such subwords start at x and y and that $n, m \le T(h, \ell)$. Also recall the formula

$$\chi^h_{\mathrm{last}}(x, y) = x < y \wedge P_{\texttt{</h>}} y \wedge \forall z((x < z \wedge z < y) \to \neg P_{\texttt{</h>}} z),$$

defined in the proof of Lemma 8, which says that y is the last position of the subword of the form $\mu_h(n)$ starting at x.

We first define a formula $\varphi^{\mathrm{lit}}_{h,\ell}(x)$ such that if the subword of γ starting at x is the encoding of a literal, then it is satisfied by α. We let

$$\begin{aligned}\psi^{\mathrm{lit}}_{h,\ell}(x) = {} & \exists y \exists x' \exists y' (P_{\texttt{<val>}} y \wedge \chi_{h,\ell}(Sx, Sy) \wedge \chi^h_{\mathrm{last}}(Sx, x') \wedge \chi^h_{\mathrm{last}}(Sy, y') \\ & \wedge (P_{+} Sx' \leftrightarrow P_{\texttt{true}} Sy')).\end{aligned}$$

Suppose that the encoding of the literal $(\neg)X_i$ starts at x. The formula $\psi^{\Gamma}_{h,\ell,0}(x)$ looks for a y such that the encoding of a pair $(j, \alpha(X_j))$ starts at y, then compares i and j, and if they are equal, checks that the symbol indicating the sign of the literal is + if, and only if, $\alpha(X_j) = \mathrm{TRUE}$. Next, we define a formula $\varphi^{\mathrm{clause}}_{h,\ell}(x)$ such that if the subword of γ starting at x is the encoding of a clause, then it is satisfied by α. We let

$$\psi^{\mathrm{clause}}_{h,\ell}(x) = \exists y (\forall z((x < z \wedge z \le y) \to \neg P_{\texttt{</clause>}} z) \wedge P_{\texttt{<lit>}} y \wedge \psi^{\mathrm{lit}}_{h,\ell}(y)).$$

It simply says that there is a position y which is still within the boundary of the clause starting at x such that a literal starts at y and this literal is satisfied. Finally, we let

$$\psi_{h,\ell}(x) = \forall y (P_{\texttt{<clause>}} y \to \psi^{\mathrm{clause}}_{h,\ell}(y)).$$

This formula says that all clauses and thus the whole CNF-formula are satisfied. □

For reasons that will become clear in the next section, we will also have to encode tuples $(\gamma, V_1, \ldots, V_t)$, where $\gamma \in \mathrm{CNF}(n)$ and $V_1, \ldots, V_t$ is a partition of $\{1, \ldots, n\}$. We add symbols $\texttt{V1}, \dots, \texttt{Vt}$ to the alphabet. So now our alphabet depends on the two parameters h and t. For every $i \in \{0, \dots, n-1\}$ and $1 \le j \le t$ we let $\mathrm{part}(i) = \texttt{Vj}$ if $X_i \in V_j$. Then we let

$$\begin{aligned}\mu_h(\gamma, V_1, \dots, V_t) = \mu_h(\gamma)\texttt{<asn>} \\ &\texttt{<val>}\ \mu_h(0)\ \mathrm{part}(0)\ \texttt{</val>} \\ &\dots \\ &\texttt{<val>}\ \mu_h(n-1)\ \mathrm{part}(n-1)\ \texttt{</val>} \\ \texttt{</asn>}.\end{aligned}$$

Even in the case $t = 1$ it will be useful to work with the encoding $\mu_h(\gamma, \{0, \dots, n-1\})$ instead of just $\mu_h(\gamma)$, because the word $\mu_h(\gamma, \{0, \dots, n-1\})$ already provides the "infrastructure" for an assignment. For brevity, we write $\mu_h(\gamma, \star)$ instead of $\mu_h(\gamma, \{0, \dots, n-1\})$.

5. Satisfiability testing through model-checking

In this section, we prove Theorem 1.

5.1. Monadic second-order logic

Theorem 11. *Assume that* PTIME $\neq$ NP. *Let $h \in \mathbb{N}$ and p a polynomial. Then there is no algorithm for* MC(MSO, $\mathbb{W}$) *whose running time is bounded by*

$$T(h, k) \cdot p(n).$$

As usual, k denotes the size of the input sentence and n the size of the input word.

Proof. Suppose that there is an algorithm $\boldsymbol{A}$ for MC(MSO, $\mathbb{W}$) whose running time is bounded by

$$T(h, k) \cdot p(n),$$

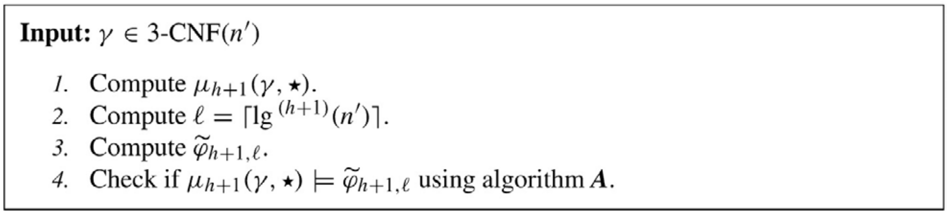
for some $h \in \mathbb{N}$ and polynomial p.

We shall prove that the satisfiability problem for 3-CNF-formulas is in polynomial time, which, by contradiction, proves the theorem. For all $\ell \in \mathbb{N}$, let

$$\widetilde{\varphi}_{h+1,\ell} = \exists X(\forall x(Xx \to P_{\texttt{V1}}x) \wedge \varphi'_{h+1,\ell}),$$

where $\varphi'_{h+1,\ell}$ is the formula obtained from the formula $\varphi_{h+1,\ell}$ of Lemma 10 by replacing the subformula $P_{\texttt{true}}Sy'$ by XSy'. Recall that $P_{\texttt{true}}Sy'$ is the only subformula of $\varphi_{h+1,\ell}$ that involves either $P_{\texttt{true}}$ or $P_{\texttt{false}}$. The subformula $\forall x(Xx \to P_{\texttt{V1}}x)$ says that X only contains elements that are at a position with symbol $\texttt{V1}$, which may simply be viewed as a placeholder for $\texttt{true}$ or $\texttt{false}$ in an assignment. The intended meaning of X is to indicate all variables set to TRUE. It is easy to see that for every $n' \le T(h+1, \ell)$ and $\gamma \in$ 3-CNF(n') we have

$$\mu_{h+1}(\gamma, \star) \models \widetilde{\varphi}_{h+1,\ell} \iff \gamma \text{ is satisfiable.} \tag{3}$$



```
Input: $\gamma \in$ 3-CNF$(n')$
    1. Compute $\mu_{h+1}(\gamma, \star)$.
    2. Compute $\ell = \lceil \lg^{(h+1)}(n') \rceil$.
    3. Compute $\widetilde{\varphi}_{h+1,\ell}$.
    4. Check if $\mu_{h+1}(\gamma, \star) \models \widetilde{\varphi}_{h+1,\ell}$ using algorithm $\boldsymbol{A}$.
```

Fig. 1.

Consider the algorithm displayed in Fig. 1, which decides if the input formula γ is satisfiable. The correctness of the algorithm follows from (3) and

$$n' = T(h+1, \lg^{(h+1)}(n')) \leq T(h+1, \lceil \lg^{(h+1)}(n') \rceil).$$

For the running time analysis, without loss of generality we can assume that $n' \leq \|\gamma\| \leq O((n')^3)$, that is, that $\|\gamma\|$ and n' are polynomially related. We claim that the running time of the algorithm is bounded by $q(n')$ for some polynomial q depending only on the fixed constant h.

Lines 1–3 of the algorithm can be implemented in time polynomial in h, n'. Recall that by Lemma 9, $|\mu_{h+1}(\gamma, \star)|$ is polynomially bounded in terms of h and n'. Thus by our assumption on the algorithm $\boldsymbol{A}$, Line 4 requires time

$$T(h, \|\widetilde{\varphi}_{h+1,\ell}\|) \cdot p(|\mu_{h+1}(\gamma, \star)|) \leq T(h, \|\widetilde{\varphi}_{h+1,\ell}\|) \cdot p'(h, n'),$$

for some polynomial p'. By Lemma 10 and the definition of $\widetilde{\varphi}_{h+1,\ell}$ we have $\|\widetilde{\varphi}_{h+1,\ell}\| \in O(h \cdot \lg h + \ell)$, that is, $\|\widetilde{\varphi}_{h+1,\ell}\| \leq c(h \cdot \lg h + \ell) \leq c(h \cdot \lg h + \lg^{(h+1)}(n') + 1)$ for some constant c. Since

$$\lim_{m \to \infty} \frac{\lg \lg m}{\lg m} = 0,$$

there is an n_0 (depending on c, h) such that for all $n' \geq n_0$ we have

$$c(h \cdot \lg h + \lg^{(h+1)}(n') + 1) \leq \lg^{(h)}(n').$$

Thus for $n' \geq n_0$ we have $T(h, \|\widetilde{\varphi}_{h+1,\ell}\|) \leq T(h, \lg^{(h)}(n')) \leq n'$. This proves the polynomial time bound. □

5.2. First-order logic

We need a few preliminaries from parameterized complexity theory. A *parameterized problem* is a set $P \subseteq \Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. 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*. A parameterized problem $P \subseteq \Sigma^* \times \mathbb{N}$ 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 $(x, k) \in P$ in time at most $f(k) \cdot p(|x|)$ steps. The class of all fixed-parameter tractable problems is denoted by FPT.

The *alternating weighted satisfiability problem* for a class Θ of propositional formulas is a parameterized version of the satisfiability problem for quantified Boolean formulas defined as follows:

AWSAT $[\Theta]$

Input: $\alpha \in \Theta$, $t \in \mathbb{N}$, a partition $V_1 \dot\cup \ldots \dot\cup V_t$ of the variables of α

Parameter: $k, t \in \mathbb{N}$

Problem: Decide if there exists a size k subset U_1 of V_1 such that for all size k subsets U_2 of V_2 there exists ... such that the truth assignment setting all variables in $U_1 \cup \ldots \cup U_t$ to TRUE and all other variables to FALSE satisfies α

The parameterized complexity class AW[$*$] is defined in terms of the alternating weighted satisfiability problem for a hierarchy of classes of propositional formulas. All we need to know here, however, is the following theorem:

Theorem 12 (Downey et al. [6], Flum and Grohe [9]). *If* AWSAT[3-CNF] *is fixed-parameter tractable then*

$$\mathrm{AW}[*] = \mathrm{FPT}.$$

We are now ready to prove our theorem:

Theorem 13. *Assume that* FPT $\neq$ AW[$*$]. *Let $h \in \mathbb{N}$ and p a polynomial. Then there is no algorithm for* $\mathrm{MC}(\mathrm{FO}, \mathbb{W})$ *whose running time is bounded by*

$$T(h,k) \cdot p(n).$$

As usual, k denotes the size of the input sentence and n the size of the input word.

To prove this theorem, we will use the following alternative characterization of fixed-parameter tractability. A parameterized problem $P \subseteq \Sigma \times \mathbb{N}$ is *eventually in polynomial time* if there is a computable function f and an algorithm, whose running time is polynomial in $|x|$ that, given an instance $(x,k) \in \Sigma^* \times \mathbb{N}$ of P with $|x| \geq f(k)$ correctly decides if $(x,k) \in P$. (The behaviour of the algorithm on instances $(x,k) \in \Sigma^* \times \mathbb{N}$ with $|x| < f(k)$ is irrelevant.)

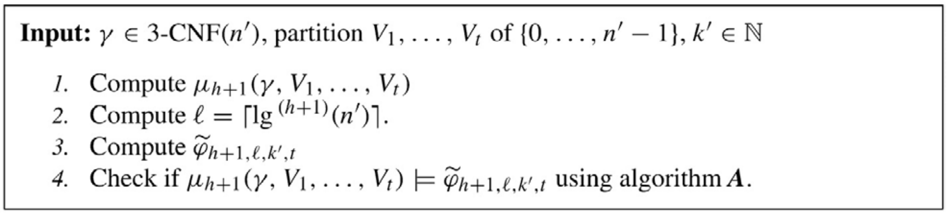
Lemma 14 (Flum and Grohe [8]). *A parameterized problem is fixed-parameter tractable if, and only if, it is computable and eventually in polynomial time.*

Proof of Theorem 13. Suppose that there is an algorithm $\boldsymbol{A}$ for $\mathrm{MC}(\mathrm{FO}, \mathbb{W})$ whose running time is bounded by

$$T(h,k) \cdot p(n),$$

for some $h \in \mathbb{N}$ and polynomial p. We shall prove that AWSAT[3-CNF] is in FPT.

For all $h, \ell, k, t \in \mathbb{N}$, let $\varphi'_{h+1,\ell,k,t}$ be the formula obtained from the formula $\varphi_{h+1,\ell}$ of Lemma 10 by replacing the (unique) subformula $P_{\mathtt{true}}Sy'$ by $\bigvee_{i=1}^{t} \bigvee_{j=1}^{k} Sy' = x_{ij}$, for



```
Input: $\gamma \in$ 3-CNF($n'$), partition $V_1, \dots, V_t$ of $\{0, \dots, n'-1\}$, $k' \in \mathbb{N}$

  1. Compute $\mu_{h+1}(\gamma, V_1, \dots, V_t)$
  2. Compute $\ell = \lceil \lg^{(h+1)}(n') \rceil$.
  3. Compute $\widetilde{\varphi}_{h+1,\ell,k',t}$
  4. Check if $\mu_{h+1}(\gamma, V_1, \dots, V_t) \models \widetilde{\varphi}_{h+1,\ell,k',t}$ using algorithm $\boldsymbol{A}$.
```

Fig. 2.

new variables x_{ij}, $1 \leq i \leq t$, $1 \leq j \leq k$. Let

$$\begin{aligned}\widetilde{\varphi}_{h+1,\ell,k,t} = \exists x_{11} \dots \exists x_{1k} \Bigg(& \bigwedge_{i=1}^{k} P_{\mathtt{V1}} x_{1i} \wedge \bigwedge_{i=1}^{k-1} x_{1i} < x_{1(i+1)} \wedge \\ & \forall x_{21} \dots \forall x_{2k} \Bigg(\Bigg(\bigwedge_{i=1}^{k} P_{\mathtt{V2}} x_{2i} \wedge \bigwedge_{i=1}^{k-1} x_{2i} < x_{2(i+1)} \Bigg) \to \\ & \vdots \\ & Q x_{t1} \dots Q x_{tk} \Bigg(\Bigg(\bigwedge_{i=1}^{k} P_{\mathtt{Vt}} x_{ti} \wedge \bigwedge_{i=1}^{k-1} x_{ti} < x_{t(i+1)} \Bigg) \overset{\wedge}{\to} \varphi'_{h+1,\ell,k,t} \Bigg) \cdots \Bigg).\end{aligned}$$

Here Q is $\forall$ if t is even and $\exists$ otherwise. Moreover, $\overset{\wedge}{\to}$ represents $\to$ if t is even and $\wedge$ if t is odd.

Then for every $n \leq T(h+1, \ell)$, $\gamma \in$ 3-CNF(n), $k \in \mathbb{N}$, and for every partition $V_1, \dots, V_t$ of $\{0, \dots, n-1\}$ we have

$$\begin{aligned}&\mu_{h+1}(\gamma, V_1, \dots, V_t) \models \widetilde{\varphi}_{h+1,\ell,k,t} \iff (\gamma, V_1, \dots, V_t) \\ &\quad \text{with parameters } (k, t) \text{ is a ‘yes’-instance AWSAT[3-CNF].} \qquad (4)\end{aligned}$$

To see this, note that the first line of $\widetilde{\varphi}_{h+1,\ell,k,t}$ says “there exists a subset $U_1 = \{x_{11}, \dots, x_{1k}\}$ of V_1 of size k” (the inequalities are used to make sure that the x_{1j} are distinct). The second line says “for all subsets $U_2 = \{x_{21}, \dots, x_{2k}\}$ of V_2 of size k”, etc. Finally, by Lemma 10, the formula $\varphi'_{h+1,\ell,k,t}$ in the last line of $\widetilde{\varphi}_{h+1,\ell,k,t}$ says that γ is satisfied if precisely the variables in $U_1 \cup \dots \cup U_t$ are set to TRUE.

Consider the algorithm displayed in Fig. 2. The correctness of the algorithm follows from (4) and

$$n' = T(h+1, \lg^{(h+1)}(n')) \leq T(h+1, \lceil \lg^{(h+1)}(n') \rceil).$$

For the running time analysis, without loss of generality we assume that $n' \leq \|\gamma\| \leq O((n')^2)$. We claim that if n' is sufficiently large, then the running time of the algorithm is bounded by $q(n')$ for some polynomial q. More precisely, we claim that there is a polynomial q and an $n_0 \in \mathbb{N}$, which is computable from h, k', t, such that for $n' \geq n_0$ the running time of the algorithm is bounded by $q(n')$. Since h is fixed and since AWSAT[3-CNF] is computable, by Lemma 14 this implies that AWSAT[3-CNF] is in FPT.

Lines 1–3 of the algorithm can be implemented in time polynomial in h, n'. By our assumption on the algorithm $\boldsymbol{A}$, Line 4 requires time

$$T(h, \|\widetilde{\varphi}_{h+1,\ell,k',t}\|) \cdot p(n) = T(h, \|\widetilde{\varphi}_{h+1,\ell,k',t}\|) \cdot p'(h, n'),$$

for some polynomial p', because $n = |\mu_{h+1}(\gamma, V_1, \dots, V_t)|$ is polynomially bounded in terms of n' and h. Since we only replace one subformula $P_{\mathtt{true}}Sy'$ by the disjunction $\bigvee_{i=1}^{t} \bigvee_{j=1}^{k} Sy' = x_{ij}$, we have

$$\|\widetilde{\varphi}_{h+1,\ell,k',t}\| \in p''(h, k', t) + O(\ell)$$

for a suitable polynomial p''. Using a similar argument as in the proof of Theorem 11, we can now derive that there is a computable n_0 depending on h, k', t such that for all $n' \geq n_0$ we have

$$T(h, \|\widetilde{\varphi}_{h+1,\ell,k',t}\|) \leq T(h, \lg^{(h)}(n')) \leq n'.$$

This proves our claim that if n' is sufficiently large, then the running time of the algorithm is bounded by $q(n')$ for some polynomial q and thus the theorem. □

Remark 15. For readers familiar with least fixed-point logic, let us point out that with the same techniques it can be proved that there is no model-checking algorithm for *monadic least fixed-point* logic on words whose running time is bounded by $T(h, k) \cdot p(n)$, for any $h \in \mathbb{N}$ and polynomial p, *under the weaker assumption that* AW[P] $\neq$ FPT.

AW[P] is a parameterized complexity class that contains AW[$*$]. A complete problem for AW[P] is the alternating weighted satisfiability problem for arbitrary Boolean circuits (as opposed to bounded depth circuits for AW[$*$]).

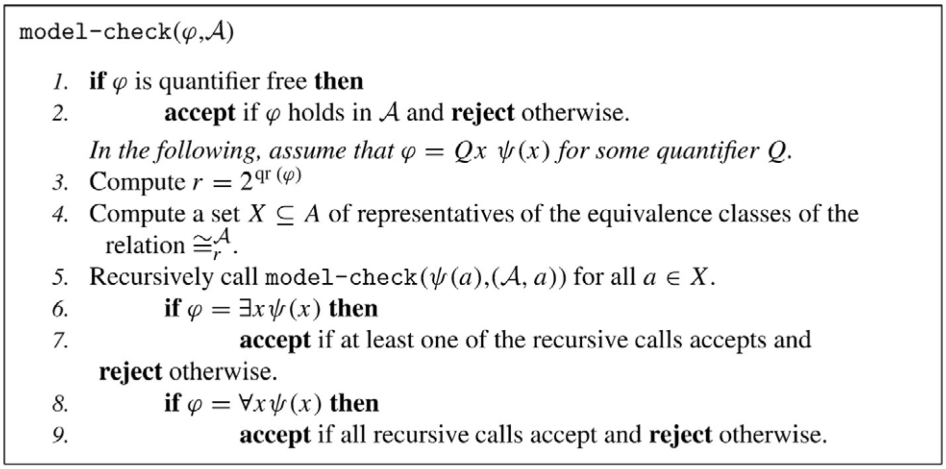
6. First-order model-checking on structures of bounded degree

In this and the next section, we investigate the parameterized complexity of first-order model-checking over structures of bounded degree. Let $\mathcal{A}$ be a τ-structure for some vocabulary τ. We call two elements $a, b \in A$ *adjacent* if they are distinct and there is an $R \in \tau$, say, r-ary, and a tuple $a_1 \dots a_r \in R^{\mathcal{A}}$ such that $a, b \in \{a_1, \dots, a_r\}$. The *degree* of an element $a \in A$ in the structure $\mathcal{A}$ is the number of elements adjacent to a, and the degree of $\mathcal{A}$ is the maximum degree of its elements. For $d \geq 1$, we denote the class of all structures of degree at most d by $\mathbb{D}(d)$.

Theorem 16 (Seese [16]). *Let $d \geq 1$. Then there is a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm solving* MC(FO, $\mathbb{D}(d)$) *in time $f(k, d) \cdot n$, where, as usual, k denotes the size of the input sentence and n the size of the input structure.*

It is quite easy to derive from Seese's proof a triply-exponential upper bound on f for a *non-uniform* version of this theorem, stating that for every fixed first-order sentence φ there is a triply exponential function f and an algorithm checking whether a given structure $\mathcal{A}$ of degree at most d satisfies φ. We shall prove a uniform version of this result, which has the additional benefit that our algorithm is quite simple.

The crucial idea, which has also been explored by Seese, is to use the locality of first-order logic. Without loss of generality we assume that vocabularies only contain relation



```
model-check($\varphi$,$\mathcal{A}$)

if $\varphi$ is quantifier free then
        accept if $\varphi$ holds in $\mathcal{A}$ and reject otherwise.
In the following, assume that $\varphi = Qx\ \psi(x)$ for some quantifier $Q$.
Compute $r = 2^{\mathrm{qr}(\varphi)}$
Compute a set $X \subseteq A$ of representatives of the equivalence classes of the
  relation $\cong_r^{\mathcal{A}}$.
Recursively call model-check($\psi(a)$,$(\mathcal{A}, a)$) for all $a \in X$.
        if $\varphi = \exists x \psi(x)$ then
                accept if at least one of the recursive calls accepts and
  reject otherwise.
        if $\varphi = \forall x \psi(x)$ then
                accept if all recursive calls accept and reject otherwise.
```

Fig. 3.

and constant symbols. (Functions can easily be simulated by relations.) We need some additional notation. A *path* of length l is a sequence of vertices $a_0, \dots, a_l \in A$ such that $a_{i-1}, a_i, i = 1, \dots, l$ are adjacent in $\mathcal{A}$. The distance between two elements $a, b \in A$ of the universe is 0, if $a = b$ and r, if the shortest path between a and b has length r. Let $r \geq 1$ and $a \in A$. The r*-neighbourhood* of a in $\mathcal{A}$, denoted by $N_r^{\mathcal{A}}(a)$ is the set of $b \in A$ such that a, b have distance at most r. Let $\mathcal{N}_r^{\mathcal{A}}(a)$ denote the substructure induced by $\mathcal{A}$ on $N_r^{\mathcal{A}}(a)$. For elements a, b of a structure $\mathcal{A}$ we write $a \cong_r^{\mathcal{A}} b$ if there is an isomorphism from $\mathcal{N}_r^{\mathcal{A}}(a)$ to $\mathcal{N}_r^{\mathcal{A}}(b)$ that maps a to b.

Recall that $\mathrm{qr}(\varphi)$ denotes the quantifier-rank of a formula φ.

Lemma 17 ([11,13]). *For every first-order formula* $\varphi(x)$ *there is an* $r \geq 1$ *such that for every structure* $\mathcal{A}$ *and* $a, b \in A$ *we have* $(a \cong_r^{\mathcal{A}} b \Longrightarrow (\mathcal{A} \models \varphi(a) \Longleftrightarrow \mathcal{A} \models \varphi(b)))$. *Furthermore,* r *can be chosen to be* $2^{\mathrm{qr}(\varphi)}$.

Fig. 3 displays a recursive model-checking algorithm for first-order sentences in prenex normal form that is based on Lemma 17. Since we can easily transform arbitrary first-order sentences into sentences in prenex normal form (algorithmically, this can be done in linear time), this also gives us an algorithm for arbitrary sentences.

Note that in the recursive calls `model-check`$(\psi(a), (\mathcal{A}, a))$ of the algorithm, we replace all occurrences of x in ψ by a new constant symbol which is interpreted by the element $a \in A$ and check if this new sentence holds in the expanded structure $(\mathcal{A}, a)$. The correctness of the algorithm follows from an easy induction on the structure of the input formula φ applying Lemma 17 in each step. Note that this algorithm works for arbitrary input structures $\mathcal{A}$.

Theorem 18. *The algorithm* `model-check` *(displayed in Fig. 3) decides* $\mathrm{MC}(\mathrm{FO}, \mathbb{D}(2))$ *in time*

$$2^{2^{O(k)}} \cdot n,$$

and MC(FO, $\mathbb{D}(d)$) *for* $d \geq 3$ *in time*

$$2^{2^{\lg d \cdot 2^{O(k)}}} \cdot n,$$

where as usual k denotes the size of the input sentence and n the size of the input structure.

Proof. We denote the running time of `model-check`$(\varphi, \mathcal{A})$ by $R(n, p, q)$, where $n = \|\mathcal{A}\|$, $q = \mathrm{qr}(\varphi)$, and p is the size of the quantifier-free part of φ. Note that $p + q \leq k (= \|\varphi\|)$. Let $r = r(q) = 2^q$,

$$s(q) = \max_{a \in A, \mathcal{A} \in \mathcal{C}} \|\mathcal{N}_r^{\mathcal{A}}(a)\|,$$

the maximal size of an r-neighbourhood, and let $t(q)$ denote the number of equivalence classes of $\cong_r^{\mathcal{A}}$. Note that there exist upper bounds for $s(q)$ and $t(q)$ only depending on the degree of the input structure (and not on n or φ). Remember that the degree is constant for the classes under consideration.

Now consider the algorithm displayed in Fig. 3. Line 1 only requires constant time. If Line 2 is executed, it requires time $O(p \cdot n)$, and the algorithm stops. Otherwise, it proceeds to Line 3, which can be executed in constant time. To execute Line 4, we maintain a list of pairs $(\mathcal{N}_r^{\mathcal{A}}(a), a)$ such that no induced substructure $(\mathcal{N}_r^{\mathcal{A}}(a), a)$ occurs twice. The size of this list never exceeds $t(q)$, hence for each a in turn, we simply compute the induced substructure, and look if it is already in the list. This requires time $O(n \cdot f(s(q)) \cdot t(q))$, if we denote the time to check isomorphism of structures of size m by $f(m)$. The loop in Lines 5–9 requires time

$$O(t(q)) + t(q) \cdot R(n, p, q - 1).$$

Putting everything together, we obtain the following recurrence for R:

$$\begin{aligned} R(n, p, 0) &\leq c_1 \cdot p \cdot n \\ R(n, p, q) &\leq c_2 \cdot n \cdot f(s(q)) \cdot t(q) + t(q) R(n, p, q - 1) \qquad (\text{for } q \geq 1), \end{aligned}$$

for suitable constants c_1, c_2. To solve this equation, we use the following simple lemma:

Lemma 19. *Let $F, g, h : \mathbb{N} \to \mathbb{N}$ such that*

$$\begin{aligned} F(0) &\leq g(0) \\ F(m) &\leq g(m) + h(m) \cdot F(m - 1) \end{aligned}$$

for all $m \in \mathbb{N}$. Then

$$F(m) \leq \sum_{i=0}^{m} g(i) \cdot \prod_{j=i+1}^{m} h(j)$$

for all $m \in \mathbb{N}$.

The lemma can be proved by a straightforward induction on q.

Applied to our function R, the lemma yields

$$\begin{aligned} R(n,p,q) &\le c_1 \cdot p \cdot n \cdot \prod_{j=1}^{q} t(j) + \sum_{i=1}^{q} c_2 \cdot n \cdot f(s(i)) \cdot t(i) \cdot \prod_{j=i+1}^{q} t(j) \\ &\le \prod_{j=1}^{q} t(j) \left(c_1 \cdot p \cdot n + \sum_{i=1}^{q} c_2 \cdot n \cdot f(s(i)) \right). \end{aligned}$$

Degree 2: The size of an r-neighbourhood in a structure $\mathcal{A} \in \mathbb{D}(2)$ is at most $2r+1$. Thus

$$s(q) \le 2^{O(q)} \le 2^{O(k)}.$$

To give an upper bound on $t(q)$, we have to take into account the number u of symbols in the vocabulary. Since we only have to consider symbols that actually appear in φ, we can assume that $u \le k$. Moreover, without loss of generality we can assume that the vocabulary only contains unary and binary relation symbols (because we are considering structures of degree 2).

Let us count the number of isomorphism types of an m-vertex structure $\mathcal{B}$ of degree 2 whose vocabulary contains u_1 unary relation symbols and u_2 binary relation symbols. The unary relations can take at most $2^{u_1 \cdot m}$ different values. There are at most m pairs of elements which can be connected by a binary relation, thus the binary relations can take at most $2^{u_2 \cdot m}$ different values. Thus the overall number of isomorphism types is bounded by $2^{(u_1+u_2)m}$.

Our r-neighbourhoods have size at most $2r+1$, so we obtain

$$t(q) \le 2^{O(k \cdot r)} = 2^{O(k \cdot 2^q)}.$$

Thus

$$\prod_{j=1}^{q} t(j) \le \prod_{j=1}^{q} 2^{O(k \cdot 2^j)} \le 2^{O(k \sum_{j=1}^{q} 2^j)} \le 2^{2^{O(k)}}.$$

Since isomorphism of structures of degree 2 can be decided in polynomial time, we obtain

$$\left(c_1 \cdot p \cdot n + \sum_{i=1}^{q} c_2 \cdot n \cdot f(s(i)) \cdot t(i) \right) \le O(2^{2^{O(k)}} \cdot n)$$

and thus

$$R(n,p,q) \le 2^{2^{O(k)}} \cdot n.$$

Degree at least 3: The calculations are similar in this case, the only important difference being that an r-neighbourhood may be of size $\Theta(d^r)$ and thus doubly exponential in q, which yields a triply exponential bound for R. □

7. Lower bounds for first-order model-checking on structures of bounded degree

In this subsection we prove lower bounds for first-order model-checking on two particularly simple classes of structures of degree two and three, respectively: The class of *words without order* and the class of *ordered binary trees*.

7.1. Words without order

Formally, a word without order over an alphabet Σ is a reduct of a word over Σ to the vocabulary $\tau_S(\Sigma) = \tau(\Sigma)\backslash\{\leq\}$. We denote the class of all words without order by $\mathbb{S}$. *Since we will only consider words without order in the following, for simplicity we often just refer to them as words.*

In this section we will only work with the encoding μ_1 (recall the definition from Section 3), but we need a refined version of Lemma 8 for $h = 1$:

Lemma 20. *Let $\ell \geq 1$ and let $\Sigma \supseteq \Sigma_1$. There is a first-order formula $\chi_\ell(x, y)$ of vocabulary $\tau_S(\Sigma_1)$ and size $O(\ell)$ such that for all words without order $\mathcal{W} \in \Sigma^*$, $a, b \in W$, and $m, n \in \{0, \dots, 2^{2^\ell}\}$ the following holds:*

If a is the first position of a subword $\mathcal{U} \sqsubseteq \mathcal{W}$ with $\mathcal{U} \cong \mu_1(m)$ and b is the first position of a subword $\mathcal{V} \sqsubseteq \mathcal{W}$ with $\mathcal{V} \cong \mu_1(n)$, then

$$\mathcal{W} \models \chi_\ell(a, b) \Longleftrightarrow m = n.$$

Furthermore, the formula χ_ℓ can be computed from ℓ in time $O(\ell)$.

Note that Lemma 8 only provides a formula $\chi_{1,l}(x, y)$ that works for $m, n \leq 2^\ell$.

Before we prove the lemma, we define a few basic formulas and notations that we need in dealing with words without order. Let $\psi(x, y)$ be a formula. For a structure $\mathcal{A}$, elements $a, b \in A$, and $\ell \geq 0$, a *ψ-path of length ℓ from a to b* is a sequence $a_0, a_1, \dots, a_\ell$ of elements of A such that $a_0 = a$, $a_\ell = b$, and $\mathcal{A} \models \psi(a_i, a_{i+1})$ for $0 \leq i < \ell$. We let $b -_\psi a$ be the minimum length of a ψ-path from a to b if there is such a path. If there is no ψ-path from a to b, we let $b -_\psi a = \infty$.

Lemma 21. *Let $\ell \geq 1$ and $\psi(x, y)$ a first-order formula.*

(1) *There exists a first-order formula $\beta_\ell^\psi(x_1, x_2)$ of size $O(\ell)$ such that for every structure $\mathcal{A}$ and all $a_1, a_2 \in A$,*

$$\mathcal{A} \models \beta_\ell^\psi(a_1, a_2) \Longleftrightarrow a_2 -_\psi a_1 \leq 2^\ell.$$

(2) *There exists a first-order formula $\delta_\ell^\psi(x_1, x_2, y_1, y_2)$ of size $O(\ell)$ such that for every structure $\mathcal{A}$ and all elements $a_1, a_2, b_1, b_2 \in A$,*

$$\mathcal{A} \models \delta_\ell^\psi(a_1, a_2, b_1, b_2) \Longleftrightarrow a_2 -_\psi a_1 \leq 2^\ell \wedge a_2 -_\psi a_1 = b_2 -_\psi b_1.$$

Proof. We only prove (2); the proof of (1) is similar, but simpler. We let

$$\delta_0^\psi(x_1, x_2, y_1, y_2) = (x_1 = x_2 \wedge y_1 = y_2) \\ \vee \big(\neg x_1 = x_2 \wedge \neg y_1 = y_2 \wedge \psi(x_1, x_2) \wedge \psi(y_1, y_2)\big),$$

and for $\ell \geq 1$

$$\begin{aligned}\delta^{\psi}_{\ell}(x_1, x_2, y_1, y_2) = \delta^{\psi}_{0}(x_1, x_2, y_1, y_2) &\\ \vee \exists x_3 \exists y_3 \forall x \forall x' \forall y \forall y' \Big(&\\ \big((x = x_1 \wedge x' = x_3 \wedge y = y_1 \wedge y' = y_3) &\\ \vee (x = x_3 \wedge x' = x_2 \wedge y = y_3 \wedge y' = y_2)\big) &\\ \to \delta^{\psi}_{\ell-1}(x, x', y, y')\Big). \quad \square &\end{aligned}$$

Proof of Lemma 20. We let $\psi(x, y) = (\neg P_{</1>}x \wedge Sx = y) \vee (P_{</1>}x \wedge x = y)$ and

$$\chi_{\ell}(x, y) = \forall x' \forall y' (\delta^{\psi}_{\ell}(x, x', y, y') \to ((P_0 x' \leftrightarrow P_0 y') \wedge (P_1 x' \leftrightarrow P_1 y'))),$$

where δ^{ψ}_{ℓ} is taken from Lemma 21(2). $\square$

Recall that 3-CNF(n) denotes the set of all formulas in 3-conjunctive normal form whose variables are among $X_0, \ldots, X_{n-1}$ and that $A(n)$ denotes the set of all truth-value assignments to these variables. Recall further the encodings of propositional formulas introduced in Section 4.

Lemma 22. *For all $l \in \mathbb{N}$ there is a first-order sentence φ_l of size $O(l)$ such that for all $n \leq 2^{2^l}$ and $(\gamma, \alpha) \in$ 3-CNF$(n) \times A(n)$ we have $\mu_1(\gamma, \alpha) \models \varphi_l \Longleftrightarrow \alpha \models \gamma$. Furthermore, φ_l can be computed in time $O(l)$.*

Proof. Recall the proof of Lemma 10. Instead of the formula $\chi_{h,\ell}$ we now use χ_{ℓ} of Lemma 20. We have to eliminate all occurrences of the order symbol $<$, which is used in the formulas $\chi^{h}_{\text{last}}(x, y)$ and $\psi^{\text{clause}}_{h,\ell}$.

Observe that the length of an encoding $\mu_1(n)$ for an $n \leq 2^{2^{\ell}}$ is in $O(2^{\ell})$. We have seen above that we can describe subwords of length up to 2^{ℓ} by formulas of length $O(\ell)$ that only use the successor relation. Therefore, replace $\chi^{h}_{\text{last}}(x, y)$ by a formula of length $O(\ell)$ that only involves the successor relation.

Moreover, since we are only considering 3-CNF(n) formulas for $n \leq 2^{2^{\ell}}$, subwords describing clauses have length $O(\ell)$. Thus again we can replace the subformulas involving the order symbol by suitable formulas of length $O(\ell)$. $\square$

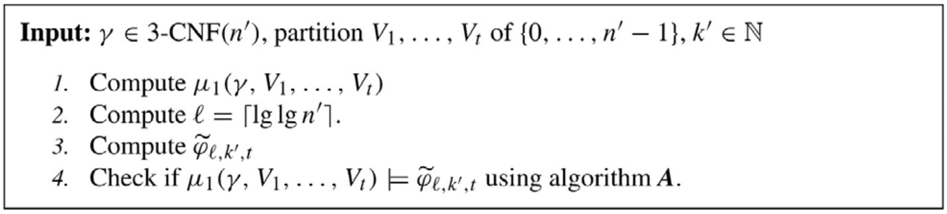
Note that the previous proof does not work for arbitrary CNF-formulas; it is crucial that the clauses have bounded length.

We are now ready to prove the main result of this section (which is Theorem 2(1)):

Theorem 23. *Assume that* FPT $=$ AW[$*$], *and let p be a polynomial. Then there is no algorithm for* MC(FO, $\mathbb{S}$) *whose running time is in*

$$2^{2^{o(k)}} \cdot p(n),$$

where k denotes the size of the input sentence and n the size of the input word.



```
Input: $\gamma \in$ 3-CNF($n'$), partition $V_1, \dots, V_t$ of $\{0, \dots, n'-1\}$, $k' \in \mathbb{N}$
  1. Compute $\mu_1(\gamma, V_1, \dots, V_t)$
  2. Compute $\ell = \lceil \lg \lg n' \rceil$.
  3. Compute $\widetilde{\varphi}_{\ell,k',t}$
  4. Check if $\mu_1(\gamma, V_1, \dots, V_t) \models \widetilde{\varphi}_{\ell,k',t}$ using algorithm $\boldsymbol{A}$.
```

Fig. 4.

Proof. Essentially, we proceed as for words with order. Suppose that there is an algorithm $\boldsymbol{A}$ for the problem MC(FO, $\mathbb{W}$) whose running time is bounded by

$$2^{2^{f(k)}} \cdot p(n),$$

for some polynomial p and a function $f(k) \in o(k)$. We shall prove that AWSAT[3-CNF] is in FPT.

For all $\ell, k, t \in \mathbb{N}$, let

$$\begin{aligned}
\widetilde{\varphi}_{\ell,k,t} = \exists x_{11} \dots \exists x_{1k} \Bigg(& \bigwedge_{i=1}^{k} P_{\mathtt{V1}} x_{1i} \wedge \bigwedge_{i=1}^{k-1} x_{1i} < x_{1(i+1)} \wedge \\
& \forall x_{21} \dots \forall x_{2k} \Bigg(\Bigg(\bigwedge_{i=1}^{k} P_{\mathtt{V2}} x_{2i} \wedge \bigwedge_{i=1}^{k-1} x_{2i} < x_{2(i+1)} \Bigg) \to \\
& \qquad \vdots \\
& Q x_{t1} \dots Q x_{tk} \Bigg(\Bigg(\bigwedge_{i=1}^{k} P_{\mathtt{Vt}} x_{ti} \wedge \bigwedge_{i=1}^{k-1} x_{ti} < x_{t(i+1)} \Bigg) \overset{\wedge}{\to} \varphi'_{\ell,k,t} \Bigg) \cdots \Bigg),
\end{aligned}$$

where $\varphi'_{\ell,k,t}$ is the formula obtained from the formula φ_ℓ of Lemma 22 by replacing the (unique) subformula $P_{\mathtt{true}} S y'$ by $\bigvee_{i=1}^{t} \bigvee_{j=1}^{k} S y' = x_{ij}$. Then for every $n \leq 2^{2^\ell}$, $\gamma \in$ 3-CNF(n), $k \in \mathbb{N}$, and for every partition $V_1, \dots, V_t$ of $\{0, \dots, n-1\}$ we have

$$\begin{aligned}
& \mu_1(\gamma, V_1, \dots, V_t) \models \widetilde{\varphi}_{\ell,k,t} \Longleftrightarrow (\gamma, V_1, \dots, V_t) \\
& \quad \text{with parameters } (k,t) \text{is a ‘yes’-instance of AWSAT[3-CNF].} \qquad (5)
\end{aligned}$$

The algorithm deciding k'-satisfiability of 3-CNF is displayed as Fig. 4.

The correctness of this algorithm follows from (5). For the analysis, without loss of generality we assume that $n' \leq \|\gamma\| \leq O((n')^2)$. We claim that if n' is sufficiently large, then the running time of the algorithm is bounded by $q(n')$ for some polynomial q. Then Lemma 14 implies that AWSAT[3-CNF] is in FPT.

Lines 1–3 of the algorithm can be done in time polynomial in n'. The crucial part is Line 4. By the assumption on algorithm $\boldsymbol{A}$ this line requires time

$$2^{2^{f(\|\widetilde{\varphi}_{\ell,k'}\|)}} \cdot p(n),$$

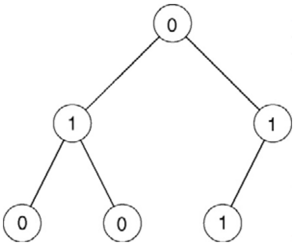


Fig. 5. The tree $\nu(38)$.

where $n = |\mu_1(\gamma, \star)|$ is polynomial in n'. It follows from Lemma 22 that

$$\|\widetilde{\varphi}_{\ell,k'}\| \leq p'(k', t) + c \cdot \ell.$$

for some polynomial p' and constant c. Hence for sufficiently large n' we have $\|\widetilde{\varphi}_{l,k'}\| \leq c' \lg \lg n'$, say, for $c' = 2c$. Since $f(k) \in o(k)$, there is an n_0 such that for all $n' \geq n_0$ we have $f(c' \lg \lg n') \leq \lg \lg n'$ and thus

$$2^{2^{f(\|\widetilde{\varphi}_{l,k'}\|)}} \leq 2^{2^{f(c' \lg \lg n')}} \leq 2^{2^{\lg \lg n'}} \leq n'.$$

This gives us the desired upper bound on the running time of our algorithm. □

7.2. Ordered binary trees

We view *ordered binary trees* as $\{S_0, S_1\}$-structures $\mathcal{T}$, with $S_0^{\mathcal{T}}$ and $S_1^{\mathcal{T}}$ being the left child and right child relations. We allow nodes to only have one child. For a finite alphabet Σ, we let $\tau_B(\Sigma) = \{S_0, S_1\} \cup \{P_s \mid s \in \Sigma\}$, where P_s, for $s \in \Sigma$, is a unary relation symbol. An ordered binary tree *over* Σ is a $\tau_B(\Sigma)$-structure whose τ-reduct is an ordered binary tree in which each vertex is contained in precisely one $P_s^{\mathcal{T}}$, for $s \in \Sigma$. We denote the class of all ordered binary trees over some finite alphabet by $\mathbb{B}$. For a node a of a tree $\mathcal{T} \in \mathbb{B}$ and $d \geq 1$, the *depth d subtree below a* is the subtree of $\mathcal{T}$ whose nodes are all descendants of a of distance at most d from a.

To proceed as in the word cases, we will encode natural numbers by trees and provide "short" formulas allowing to compare "large" encoded numbers. For $\ell \in \mathbb{N}$, let $\mathcal{T}_\ell$ be the ordered binary tree with vertex set $\{0, \ldots, \ell\}$ and root 0 in which the children of i are $2i+1$ and $2i + 2$. Recall that $L(n)$ denotes the length of the binary encoding of $n \in \mathbb{N}$. We let $\nu(n)$ be the ordered binary tree over $\{\mathtt{0}, \mathtt{1}\}$ whose underlying tree is $\mathcal{T}_{L(n)}$ and in which, for $i = 0, 1$,

$$P_i^{\mathcal{T}(n)} = \{j \leq L(n) \mid \mathrm{bit}(j, n) = i\}.$$

Example 24. Fig. 5 shows the encoding of 38, the binary representation of which is 100110.

The next lemma corresponds to Lemmas 8 and 20.

Lemma 25. *Let $\ell \geq 1$. There is a formula $\chi_\ell(x, y)$ of vocabulary $\tau_B(\{0, 1\})$ of size $O(\ell)$ such that for all ordered binary trees $\mathcal{T} \in \mathbb{B}$, $a, b \in T$ and $m, n \in \{0, \dots, 2^{2^{2^l}}\}$ the following holds:*

If the depth 2^ℓ subtree below a is isomorphic to $\nu(n)$ and the depth 2^ℓ subtree below b is isomorphic to $\nu(m)$ then

$$\mathcal{T} \models \chi_\ell(a, b) \Longleftrightarrow m = n.$$

Furthermore $\chi_\ell(x, y)$ can be computed in time $O(\ell)$.

Proof. We construct a formula $\chi_\ell(x, y)$ characterizing depth 2^ℓ subtrees up to isomorphism. This formula identifies binary encodings of length up to 2^{2^ℓ}, which proves the claim. We proceed as in the proof of Lemma 21. First, we say that to go from vertices x_1 to x_2 and from y_1 to y_2 we must follow the same sequence of S_0, S_1-successors. Let

$$\begin{aligned}\psi_0(x_1, x_2, y_1, y_2) = {} & (S_0x_1x_2 \wedge S_0y_1y_2)\\ & \vee (S_1x_1x_2 \wedge S_1y_1y_2)\\ & \vee (x_1 = x_2 \wedge y_1 = y_2),\end{aligned}$$

and for $l \geq 1$

$$\begin{aligned}\psi_l(x_1, x_2, y_1, y_2) = {} & \exists x_3 \exists y_3 \forall x \forall x' \forall y \forall y' ((x_1 = x \wedge x_3 = x' \wedge y_1 = y \wedge y_3 = y')\\ & \vee (x_3 = x \wedge x_2 = x' \wedge y_3 = y \wedge y_2 = y')\\ & \qquad\qquad \to \psi_{l-1}(x, x', y, y')).\end{aligned}$$

Using this formula we let

$$\chi_l(x, y) = \forall x' \forall y' (\psi_l(x, x', y, y') \to ((P_1x' \leftrightarrow P_1y') \wedge (P_0x' \leftrightarrow P_0y')),$$

which is the sought formula. □

Now we proceed as before and encode formulas of 3-CNF(n) for some n as an ordered binary tree over some alphabet Σ. For $\gamma \in$ 3-CNF let $\nu(\gamma)$ be the binary tree $\mathcal{T}$ constructed as follows: let $\mathcal{W}$ be the word without order $\mu_1(\gamma)$, and consider $\mathcal{W}$ as a tree of S_1-successors without any S_0-successors. To get $\mathcal{T}$ we substitute each subword $\mathcal{U}$ of $\mathcal{W}$ of the form $\mu_1(m)$ by a single vertex v such that v's S_0-successor is the root of a copy of $\nu(m)$, while its S_1-successor is the first position after $\mathcal{U}$ in $\mathcal{W}$. v itself carries the new symbol `var`.

We extend the definition of ν to pairs $(\gamma, \alpha) \in$ 3-CNF$(n) \times A(n)$ and tuples $(\gamma, V_1, \dots, V_t)$ by applying the same substitution process. This encoding gives us the following lemma, whose proof is omitted since it resembles the proof of Lemma 10 using the newly introduced encoding ν together with the decoding formulas $\chi_\ell(x, y)$.

Lemma 26. *For all $\ell \in \mathbb{N}$ there is a first-order sentence ψ_ℓ of size $O(l)$ such that for all $n \leq 2^{2^{2^\ell}}$ and $(\gamma, \alpha) \in$ 3-CNF$(n) \times A(n)$ we have $\nu(\gamma, \alpha) \models \psi_\ell \Longleftrightarrow \alpha \models \gamma$. Furthermore, ψ_ℓ can be computed in time $O(\ell)$.*

Now we are ready to state the second main result of this section, which is Theorem 2(2). We omit the proof, which is analogous to the proof of Theorem 23.

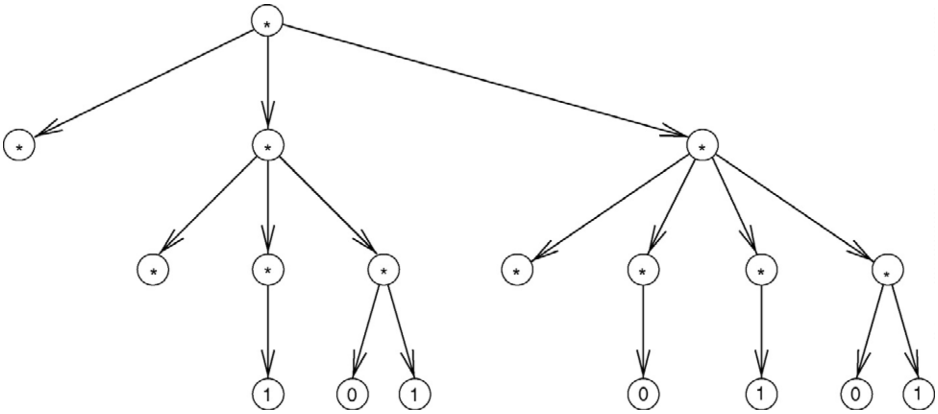


Fig. 6. The tree $\nu_3(40\,961)$.

Theorem 27. *Assume that* FPT $=$ AW$[*]$, *and let p be a polynomial. Then there is no algorithm for* MC(FO, $\mathbb{B}$) *whose running time is in*

$$2^{2^{2^{o(k)}}} \cdot p(n),$$

where k denotes the size of the input sentence and n the size of the input tree.

8. Lower bounds for first-order model-checking on trees

In this last section we prove a non-elementary lower bound for first-order model-checking over unranked trees. We need the same ingredients as before: suitable encodings of natural numbers and small formulas for comparing two numbers.

For simplicity, we work with *directed labelled trees*. In Remark 33 we describe how to get rid of labels and directed edges in order to transfer the results to plain undirected trees. But for now we view a *tree* as an $\{E\}$-structures $\mathcal{T}$ with $E^{\mathcal{T}}$ being the child-relation. For a finite alphabet Σ we let $\tau_T(\Sigma) = \{E\} \cup \{P_s \mid s \in \Sigma\}$. Then a tree *over* Σ is a $\tau_T(\Sigma)$-structure $\mathcal{T}$ whose $\{E\}$-reduct is a tree and in which each vertex is contained in precisely one $P_s^{\mathcal{T}}$, for $s \in \Sigma$. We denote the class of all trees over some alphabet by $\mathbb{T}$.

Recall that $T(h, 2)$ denotes a tower of 2s of height $h + 1$ and that $\text{bit}(i, n)$ denotes the ith bit in the binary representation of n. For every $h \geq 0$ and $n \in \{0, \ldots, T(h, 2) - 1\}$ we define $\nu_h(n)$ to be the following tree over $\{\mathtt{0}, \mathtt{1}, *\}$:

(1) If $h = 0$, we let $\nu_0(0)$ be a single node labelled by $\mathtt{0}$. Likewise, let $\nu_0(1)$ be a single node labelled by $\mathtt{1}$.

(2) If $h \geq 1$, we let $\nu_h(n)$ be the tree formed by taking a new root, labelling it by $*$, and attaching to it the tree $\nu_{h-1}(i)$ for each i such that $\text{bit}(i, n) = 1$.

Example 28. Fig. 6 shows the ν_3-encoding of $40\,961 = 2^{15} + 2^{13} + 2^0$. The tree is constructed as follows:

- To construct $\nu_3(40\,961)$, by clause (2), we take a new root labelled by $*$ and attach three trees to this root: $\nu_2(0)$, $\nu_2(13)$, $\nu_2(15)$.
- The binary representation of 0 consists of 0s only. Thus to construct $\nu_2(0)$, we take a new root labelled by $*$ and attach no children. This explains the leftmost leaf labelled $*$.
- We have $13 = 2^0 + 2^2 + 2^3$. Thus to construct $\nu_2(13)$, we take a new root labelled by $*$ and attach three children labelled $\nu_1(0)$, $\nu_1(2)$, and $\nu_1(3)$.
- $\nu_1(0)$ is again a tree consisting of just one node labelled $*$. This explains the second leaf labelled $*$.
- We have $2 = 2^1$. Thus to construct $\nu_1(2)$, we take a new root labelled by $*$ and attach one child labelled by $\nu_0(1)$.
- $\nu_0(1)$ is the 1-node tree labelled $\mathtt{1}$.
- The remaining subtrees are constructed similarly.

Lemma 29. *There is an algorithm that, given h and $n \in \{0, \dots, T(h, 2)\}$, computes $\nu_h(n)$ in time $O(h \cdot \lg^2 n)$. Furthermore, $|\nu_h(n)| \in O(h \cdot \lg^2 n)$.*

Proof. A simple recursive procedure will do. The running time analysis uses the same ideas as the proofs of Lemmas 6 and 7. □

The next lemma corresponds to Lemmas 8 and 20.

Lemma 30. *Let $h \geq 1$. There is a first-order formula $\xi_h(x, y)$ of size $O(h)$ such that for all trees $\mathcal{T}$ over Σ, $a, b \in T$, and $m, n \in \{0, \dots, T(h, 2) - 1\}$ the following holds:*

If the subtrees of $\mathcal{T}$ rooted at a, b are isomorphic to $\nu_h(m)$ and $\nu_h(n)$, respectively, then $\mathcal{T} \models \chi_h(a, b)$ if, and only if, $m = n$.

Proof. $\xi_0(x, y)$ simply is the formula $P_0 x \leftrightarrow P_0 y$. Let $\xi_h(x, y)$ already be defined. $\xi_{h+1}(x, y)$ says that for each successor x_1 of x there is a successor y_1 of y such that $\xi_h(x_1, y_1)$ and vice versa. As usual, we have to take care to avoid duplication of the subformula ξ_h. We let

$$\begin{aligned}\xi_{h+1}(x, y) = \forall z_1((Exz_1 \vee Eyz_1) \rightarrow \exists z_2((Exz_1 \rightarrow Eyz_2)\\ \wedge (Eyz_1 \rightarrow Exz_2) \wedge \xi_h(z_1, z_2))),\end{aligned}$$

which has the intended meaning and the desired size. □

We encode 3-CNF-formulas as trees over a suitable alphabet Σ in essentially the same way we did with binary trees in Section 7.2, using the encoding ν_h instead of ν. Then for every h we get an encoding ν_h of formulas in 3-CNF(n) for $n < T(h, 2)$. We extended the definition of ν_h to pairs $(\gamma, \alpha) \in$ 3-CNF$(n) \times A(n)$ and to tuples $(\gamma, V_1, \dots, V_\ell)$.

Lemma 31. *For all $h \in \mathbb{N}$ there is a first-order sentence ζ_h of size $O(h)$ such that for all $n < T(h, 2)$ and $(\gamma, \alpha) \in$ 3-CNF$\times A(n)$ we have $\nu_h(\gamma, \alpha) \models \zeta_h \Leftrightarrow \alpha \models \gamma$. Furthermore, ζ_h can be computed in time $O(h)$.*

We omit the simple proof.

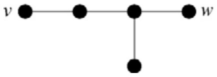
Theorem 32. *Assume that* FPT $\neq$ AW[$*$]. *Let $h \in \mathbb{N}$ and p a polynomial. Then there is no algorithm for* MC(FO, $\mathbb{T}$) *whose running time is bounded by*

$$T(h, k) \cdot p(n),$$

where k denotes the size of the input sentence and n the size of the input tree.

The proof is analogous to our earlier lower bound proofs.

Remark 33. Even though we only stated the lower bound result for labelled binary trees, it also holds for unlabelled undirected trees, that is, connected acyclic undirected graphs. To see this, we first note that the alphabet and thus the vocabulary of the formula ζ_h of Lemma 31 does not depend on h. Suppose the vocabulary of ζ_h is $\{E, P_1, \dots, P_p\}$. To get rid of the directed edges, we replace each directed edge from a vertex v to a vertex w by the following subgraph:



To get rid of the unary relations, we attach $(i + 2)$ new children to each node in P_i and delete P_i.

9. Conclusions

It is interesting to observe that the complexity-theoretic assumptions we use to prove our theorems, that is, PTIME $\neq$ NP for the theorem on MSO and FPT $\neq$ AW[$*$] for the theorems on MSO, are precisely the assumptions needed to prove that the model-checking problem for the respective logic on arbitrary structures is not FPT. It remains an open problem to weaken the complexity-theoretic assumptions to PTIME $\neq$ PSPACE. Note that PTIME $\neq$ PSPACE is a necessary assumption for all our lower bounds, because if PTIME $=$ PSPACE then model-checking for monadic second-order logic is in PTIME.

There is a significant gap between the lower bounds for model-checking on words provided by Theorem 1 and the upper bound $T(O(k), 1) \cdot n$ (a tower of 2s of height $O(k)$). It would be nice to narrow this gap, maybe by proving that there is no $T(o(k), 1) \cdot p(n)$ algorithm for first-order or monadic second-order model-checking on words.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[2] J.R. Büchi, Weak second-order arithmetic and finite automata, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 6 (1960) 66–92.

[3] B. Courcelle, Graph rewriting: an algebraic and logic approach, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, Elsevier Science Publishers, 1990, pp. 194–242.

[4] N.J. Cutland, Computability, Cambridge University Press, 1980.

[5] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer-Verlag, 1999.

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

[7] H.-D. Ebbinghaus, J. Flum, W. Thomas, Mathematical Logic, 2nd edition, Springer-Verlag, 1994.

[8] J. Flum, M. Grohe, Describing parameterized complexity classes, in: H. Alt, A. Ferreira (Eds.), Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 2285, Springer-Verlag, 2002, pp. 359–371.

[9] J. Flum, M. Grohe, Model-checking problems as a basis for parameterized intractability, Technical Report 23/2003, Fakultät für Mathematik und Physik, Albert-Ludwigs-Universität Freiburg, 2003.

[10] M. Frick, M. Grohe, Deciding first-order properties of locally tree-decomposable structures, Journal of the ACM 48 (2001) 1184–1206.

[11] W. Hanf, Model-theoretic methods in the study of elementary logic, in: J. Addison, L. Henkin, A. Tarski (Eds.), The Theory of Models, North Holland, 1965, pp. 132–145.

[12] H. Kamp, Tense Logic and the theory of linear order, Ph.D. Thesis, University of California, Los Angeles, 1968.

[13] L. Libkin, Logics with counting and local properties, ACM Transactions on Computational Logic 1 (2000) 33–59.

[14] O. Lichtenstein, A. Pnueli, Finite state concurrent programs satisfy their linear specification, in: Proceedings of the Twelfth ACM Symposium on the Principles of Programming Languages, 1985, pp. 97–107.

[15] K. Reinhardt, The complexity of translating logic to finite automata, in: E. Grädel, W. Thomas, T. Wilke (Eds.), Automata, Logics, and Infinite Games, Lecture Notes in Computer Science, vol. 2500, Springer-Verlag, 2002, pp. 235–242 (Chapter 13).

[16] D. Seese, Linear time computable problems and first-order descriptions, Mathematical Structures in Computer Science 6 (1996) 505–526.

[17] L.J. Stockmeyer, The Complexity of Decision Problems in Automata Theory, Ph.D. Thesis, Department of Electrical Engineering, MIT, 1974.

[18] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time, in: Proceedings of the 5th ACM Symposium on Theory of Computing, 1973, pp. 1–9.

[19] P. van Emde Boas, Machine models and simulations, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. 1, Elsevier Science Publishers, 1990, pp. 1–66.

[20] M.Y. Vardi, The complexity of relational query languages, in: Proceedings of the 14th ACM Symposium on Theory of Computing, 1982, pp. 137–146.