Modulo Counting on Words and Trees[*][†]

Bartosz Bednarczyk[1] and Witold Charatonik[2]

1 **Computer Science Department, ENS Paris-Saclay, Cachan, France; and Institute of Computer Science, University of Wrocław, Wrocław, Poland**
`bartosz.bednarczyk@ens-paris-saclay.fr`

2 **Institute of Computer Science, University of Wrocław, Wrocław, Poland**
`wch@cs.uni.wroc.pl`

Abstract

We consider the satisfiability problem for the two-variable fragment of the first-order logic extended with modulo counting quantifiers and interpreted over finite words or trees. We prove a small-model property of this logic, which gives a technique for deciding the satisfiability problem. In the case of words this gives a new proof of ExpSpace upper bound, and in the case of trees it gives a 2-ExpTime algorithm. This algorithm is optimal: we prove a matching lower bound by a generic reduction from alternating Turing machines working in exponential space; the reduction involves a development of a new version of tiling games.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases satisfiability, trees, words, two-variable logic, modulo quantifiers

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.12

1 Introduction

Two-variable logics. Two-variable logic, FO^2, is one of the most prominent decidable fragments of first-order logic. It is important in computer science because of its decidability and connections with other formalisms like modal, temporal and description logics or query languages. The satisfiability problem for FO^2 is NExpTime-complete and satisfiable formulas have models of exponential size. In this paper we are interested in extensions of FO^2 interpreted over finite words and trees. It is known that FO^2 over words can express the same properties as unary temporal logic [9] and FO^2 over trees is precisely as expressive as the navigational core of XPath, a query language for XML documents [16]. The satisfiability problem for FO^2 over words is shown to be NExpTime-complete in [9], and over trees – ExpSpace-complete in [4]. Recently it was shown that these complexities do not change[7, 3] if counting quantifiers of the form $\exists^{\leq k}, \exists^{\geq k}$ are added.

Modulo counting quantifiers. First-order logic has a limited expressive power. For example, it cannot express even such simple quantitative properties as parity. To overcome this problem, Wolfgang Thomas with his coauthors introduced in the 80s of the last century *modulo counting quantifiers* of the form "there exists a mod b elements x such that ...". A survey of results in first order logic extended with modulo counting quantifiers can be found in [22]. Recent research in this area involves a study of definability of regular languages on words and its connections to algebra [22], [23]; definable languages of $(\mathbb{N}, +)$ [19], [14]; equivalences

[*] Supported by the Polish National Science Centre grant No. 2016/21/B/ST6/01444.
[†] A full version of the paper is available at `http://arxiv.org/abs/1710.05582`.

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 12; pp. 12:1–12:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of finite structures [17]; definable tree languages [18], [5]; locality [11], [12]; extensions of Linear Temporal Logic [1], [21], complexity of the model-checking problem [12], [6]. Not much is known about the complexity of the satisfiability problem for this logic. The only work that we are aware of is [15], which proves ExpSpace-completeness of the satisfiability problem for FO^2 with modulo counting quantifiers over finite words. On the other hand, a simple adaptation of automata techniques for deciding WS1S or WS2S gives a non-elementary decision procedure for the satisfiability problem of full first-order logic with modulo quantifiers over words or ranked trees.

Our contribution. We consider the satisfiability problem for FO^2 with modulo counting quantifiers interpreted over finite words and trees. We provide an alternative to [15] proof of ExpSpace upper bound for the case of words. This proof is based on a small-model property of this logic and can be extended to the case of ordered and unranked trees. By ordered, we mean that the list of children of any node is ordered by the sibling relation, and by unranked, that there is no limit on the number of children. We prove that the case of such trees is 2-ExpTime-complete. For the upper bound, we again prove a small-model theorem and then give an alternating algorithm that decides the problem in exponential space. Since AExpSpace = 2-ExpTime, this gives the desired upper bound. With some obvious modifications this algorithm can be also applied to unordered or ranked trees. For the lower bound, we develop a new version of tiling games that can encode computations of alternating Turing machines working in exponential space, and can be encoded in the logic. In our encoding we do not use ordering of children, and the number of children of any node is bounded by 2, which shows that already the case of unordered and ranked trees is 2-ExpTime-hard.

Due to space limits some proofs and encodings are omitted. They can be found in the full version of the paper [2].

2 Preliminaries

Tuples and modulo remainders. By k-tuple of numbers we mean an element of the set $\mathbb{N}^k$. By $\vec{0}_k$ and $\vec{1}_k$ we will denote k-tuples consisting of, respectively, only zeros and only ones. We denote the i-th element of a k-tuple t by $\pi_i(t)$. We will often drop the subscript k if the dimension k is clear from the context.

Consider a finite set X, a number $k \in \mathbb{N}$ and a mapping f from X to $\mathbb{N}^k$. We say that f is *zero*, if for all $x \in X$ we have $f(x) = \vec{0}_k$. We say that f is a *singleton*, if there exists a unique argument $x \in X$ such that $f(x) = \vec{1}_k$ and for all other arguments the function f is zero.

Let n be a positive integer. By $\mathbb{Z}_n$ we denote the set of all remainders modulo n, that is the set $\{0, 1, \dots, n{-}1\}$. We denote by r_n the remainder function modulo n, that is $r_n : \mathbb{N} \to \mathbb{Z}_n$, $r_n(x) = x \bmod n$.

Syntax and semantics. We refer to structures with fraktur letters, and to their universes with the corresponding Roman letters. We always assume that structures have non-empty universes. We work with signatures of the form $\tau = \tau_0 \cup \tau_{nav}$, where τ_0 is a set of unary relational symbols, and $\tau_{nav} \subseteq \{\downarrow, \downarrow^+, \rightarrow, \rightarrow^+, \leq, succ\}$ is the set of *navigational* binary symbols, which will be interpreted in a special way, depending on which kind of structures, words or trees, are considered as models.

We define the two-variable fragment of first-order logic with modulo counting quantifiers $\mathrm{FO}^2_{\mathrm{MOD}}$ as the set of all first-order formulas (over the signature τ), featuring only the variables x and y, extended with modulo counting quantifiers of the form $\exists^{\leq k,l}, \exists^{=k,l}$ and $\exists^{\geq k,l}$, where $l \in \mathbb{Z}_+$ is a positive integer and $k \in \mathbb{Z}_l$. The formal semantics of such quantifiers is as follows: a formula $\exists^{\bowtie k,l}\, y\, \varphi(x,y)$, where $\bowtie \in \{\leq, =, \geq\}$, is true at element a of a structure $\mathfrak{M}$, in symbols $\mathfrak{M}, a \models \left(\exists^{\bowtie k,l}\, y\, \varphi(x,y)\right)$ if and only if $r_l\left(\,|\{b \in M : \varphi[a,b]\}|\,\right) \bowtie k$. When measuring the length of formulas we will assume binary encodings of numbers k and l in superscripts of modulo quantifiers.

In [15] the authors use a different notation for modulo counting quantifiers in their logic $\mathrm{FO}\textsc{mod}^2[<]$. It seems that the main difference is that they require equality in place of our $\bowtie$ comparison. The use of $\leq$ and $\geq$ operators makes the logic exponentially more succinct. For example the formula $\exists^{\geq 2^n, 2\cdot 2^n} x\, p(x)$ has length $\mathcal{O}(n)$, while an equivalent formula in $\mathrm{FO}\textsc{mod}^2[<]$ has the form $\bigvee_{i=2^n}^{2\cdot 2^n} \exists^{=i,2\cdot 2^n} x\, p(x)$ and its length is $\Omega(2^n \cdot n)$.

We write $\mathrm{FO}^2_{\mathrm{MOD}}[\tau_{nav}]$, to denote that the only binary symbols that are allowed in signatures are from τ_{nav}. We will work with two logics: $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, \mathit{succ}]$ and $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$. The former is interpreted over finite words, where $\leq$ denotes the linear order over word positions and *succ* is the successor relation. The latter logic is interpreted over finite, unranked and ordered trees. The interpretation of the symbols from τ_{nav} is the following: $\downarrow$ is interpreted as the child relation, $\rightarrow$ as the right sibling relation, and $\downarrow^+$ and $\rightarrow^+$ as their respective transitive closures. We read $u \downarrow w$ as "w is a *child* of u" and $u \rightarrow w$ as "w is the *right sibling* of u". We will also use other standard terminology like *ancestor*, *descendant*, *preceding-sibling*, *following-sibling*, etc. In both cases of words and trees all elements of the universe can be labeled by an arbitrary number of unary predicates from τ_0.

Normal form. As usual when working with two-variable logic, it is convenient to use so-called Scott normal form [20]. The main feature of such form is that nesting of quantifiers is restricted to depth two. Below we adapt this notion to the setting of $\mathrm{FO}^2_{\mathrm{MOD}}$.

▶ **Definition 1.** We say that a formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}$ is in *normal form*, if

$$\varphi = \forall x \forall y\, \chi(x,y) \wedge \bigwedge_{i=1}^{n} \forall x \exists y\, \chi_i(x,y) \wedge \bigwedge_{j=1}^{m} \forall x \exists^{\bowtie_j k_j, l_j} y\, \psi_j(x,y)$$

where $\bowtie_i \in \{\leq, \geq\}$, each l_j is a positive integer and each $k_j \in \mathbb{Z}_{l_j}$ and all χ, χ_i and ψ_j formulas are quantifier-free. We require both n and m parameters to be non-zero.

The following lemma is proved by a routine adaptation of a normal form proof from [10].

▶ **Lemma 2.** *Let φ be a formula from $\mathrm{FO}^2_{\mathrm{MOD}}$ over a signature τ. There exists a polynomially computable $\mathrm{FO}^2_{\mathrm{MOD}}$ normal form formula φ' over signature τ', consisting of τ and polynomially many additional unary symbols, such that φ and φ' are equisatisfiable.*

In the next sections, when a normal form formula φ is considered we always assume that it is as in Definition 1. In particular we allow ourselves, without explicitly recalling the shape of φ, to refer to its parameters n, m, components χ, χ_i, ψ_j and numbers k_j, l_j given in modulo counting quantifiers.

Consider a conjunct $\forall x \exists y\, \chi_i(x,y)$ from an $\mathrm{FO}^2_{\mathrm{MOD}}$ normal form formula φ. Let $\mathfrak{M}$ be its finite model and let $v \in M$ be an arbitrary element from the structure. Then an element $w \in M$ such that $\mathfrak{M} \models \chi_i[v,w]$ is called a *χ_i-witness* for v. Analogously, we will speak about ψ_j-witnesses for modulo conjuncts or simply witnesses if a formula is clear from the context.

Order formulas. Let us call *order formulas* the formulas specifying relative positions of pairs of elements in a structure with respect to the binary navigational predicates.

In the case of words, when $\tau_{nav} = \{\leq, succ\}$, there are five possible order formulas: $x{=}y$, $succ(x, y)$, $succ(y, x)$, $x{\neq}y \wedge \neg succ(x, y) \wedge x{\leq}y$ and $x{\neq}y \wedge \neg succ(y, x) \wedge y{\leq}x$. We denote them respectively as $\theta_{=}, \theta_{+1}, \theta_{-1}, \theta_{\ll}, \theta_{\gg}$. Let Θ_{words} be the set of these five formulas.

In the case of trees, when $\tau_{nav} = \{\downarrow, \downarrow^{+}, \rightarrow, \rightarrow^{+}\}$, we use $x{\not\sim}y$ to abbreviate the formula stating that x and y are in *free position*, i.e., that they are related by none of the navigational binary predicates available in the signature. There are ten possible order formulas: $x{\downarrow}y$, $y{\downarrow}x$, $x{\downarrow^{+}}y \wedge \neg(x{\downarrow}y)$, $y{\downarrow^{+}}x \wedge \neg(y{\downarrow}x)$, $x{\rightarrow}y$, $y{\rightarrow}x$, $x{\rightarrow^{+}}y \wedge \neg(x{\rightarrow}y)$, $y{\rightarrow^{+}}x \wedge \neg(y{\rightarrow}x)$, $x{\not\sim}y$, $x{=}y$. They are denoted, respectively, as: $\theta_{\downarrow}$, $\theta_{\uparrow}$, $\theta_{\downarrow\downarrow^{+}}$, $\theta_{\uparrow\uparrow^{+}}$, $\theta_{\rightarrow}$, $\theta_{\leftarrow}$, $\theta_{\rightrightarrows^{+}}$, $\theta_{\leftleftarrows^{+}}$, $\theta_{\not\sim}$, $\theta_{=}$. Let Θ_{trees} be the set of these ten formulas. We will use symbol Θ to denote either Θ_{words} or Θ_{trees} if the type of structure is not important or clear from context.

Types and local configurations. Following a standard terminology, an *atomic* 1-*type* over a signature τ is a maximal satisfiable set of atoms or negated atoms involving only the variable x. Usually we identify a 1-type with the conjunction of all its elements. We will denote by $\boldsymbol{\alpha}$ the set of all 1-types over τ. Note that the size of $\boldsymbol{\alpha}$ is exponential in the size of τ. For a given τ-structure $\mathfrak{M}$, and an element $v \in M$ we say that v *realizes* a 1-type α, if α is the unique 1-type such that $\mathfrak{M} \models \alpha[v]$. We denote by $\mathrm{tp}^{\mathfrak{M}}(v)$ the 1-type realized by v.

A 1-type provides us only information about a single element of a model. Below we generalize this notion such that for each element of a structure it provides us also some information about surrounding elements and their 1-types.

▶ **Definition 3.** An $(l_1, l_2, \ldots, l_m)$-full type $\overline{\alpha}$ over a signature τ is a function of the type $\overline{\alpha} : \Theta \rightarrow \boldsymbol{\alpha} \rightarrow \{0, 1\} \times \mathbb{Z}_{l_1} \times \mathbb{Z}_{l_2} \times \ldots \times \mathbb{Z}_{l_m}$ (a function that takes an order formula from Θ and returns a function that takes a 1-type from $\boldsymbol{\alpha}$ and returns a tuple of integers). Additionally, we require that all $(l_1, l_2, \ldots, l_m)$-full types satisfy the following conditions:

- $\overline{\alpha}(\theta_{=})$ is a singleton,
- $\overline{\alpha}(\theta)$ is either zero or singleton for $\theta \in \{\theta_{+1}, \theta_{-1}, \theta_{\rightarrow}, \theta_{\uparrow}, \theta_{\leftarrow}\}$ and
- if $\overline{\alpha}(\theta_{+1})$ (respectively $\overline{\alpha}(\theta_{-1}), \overline{\alpha}(\theta_{\rightarrow}), \overline{\alpha}(\theta_{\downarrow}), \overline{\alpha}(\theta_{\leftarrow}), \overline{\alpha}(\theta_{\uparrow})$) is zero then also the function $\overline{\alpha}(\theta_{\gg})$ (respectively $\overline{\alpha}(\theta_{\ll}), \overline{\alpha}(\theta_{\rightrightarrows^{+}}), \overline{\alpha}(\theta_{\downarrow\downarrow^{+}}), \overline{\alpha}(\theta_{\leftleftarrows^{+}}), \overline{\alpha}(\theta_{\uparrow\uparrow^{+}})$) is zero.

In the following, if the numbers $l_1, l_2, \ldots, l_m$ are clear from the context or not important, we will be omitting them.

Let us briefly describe the idea behind the notion of full types. Ideally, for a given element v of a structure $\mathfrak{M}$, we would like to know the exact number of occurrences of each 1-type on each relative position. However, to keep the memory usage under control, we store only a summary of this information. For the purpose of $\forall\exists$ conjuncts of a formula in normal form, it is enough to know if a type α occurs at least once in a given relative position. This information is stored in the 0-1 part of a full type. Additionally, for the purpose of $\exists^{\bowtie_j k_j, l_j}$ conjuncts, we store the remainders of the total numbers of occurrences of each 1-type α modulo each of l_j appearing in modulo quantifiers.

▶ **Definition 4.** Let $\mathfrak{M}$ be a τ-structure and v an arbitrary element from M. We will denote by $(l_1, l_2, \ldots, l_m)$–$\mathrm{ftp}^{\mathfrak{M}}(v)$ the unique $(l_1, l_2, \ldots, l_m)$-full type *realized* by v in $\mathfrak{M}$, i.e., the $(l_1, l_2, \ldots, l_m)$-full type $\overline{\alpha}$ such that for all order formulas $\theta \in \Theta$ and for all atomic 1-types α, the value of $\overline{\alpha}(\theta)(\alpha)$ is equal to the tuple $(cut_1(W), r_{l_1}(W), r_{l_2}(W), \ldots, r_{l_m}(W))$, where W is the number of occurrences of elements of type α in the relative position described by θ, namely

$$W = |\{w \in M : \mathfrak{M} \models \theta[v, w] \wedge \mathrm{tp}^{\mathfrak{M}}(w) = \alpha\}|.$$

and $cut_1(W)$ is 0 if W is empty and 1 otherwise.

The following definition and lemma are basic tools used in the proofs of small-model properties $\mathrm{FO}^2_{\mathrm{MOD}}$.

▶ **Definition 5** (φ-consistency). Let φ be a $\mathrm{FO}^2_{\mathrm{MOD}}$ formula in normal form and let α be the unique 1-type satisfying $\overline{\alpha}(\theta_=)(\alpha) = \vec{1}$. We say that a $(l_1, l_2, \ldots, l_m)$-full type $\overline{\alpha}$ is *φ-consistent*, if it satisfies the following conditions:

1. It does not violate the $\forall\forall$ subformula i.e. for all order formulas $\theta \in \Theta$ and for all 1-types β such that $\overline{\alpha}(\theta)(\beta) \neq \vec{0}$, the implication $\alpha(x) \wedge \beta(y) \wedge \theta(x, y) \models \chi(x, y)$ holds.
2. There is a witness for each $\forall\exists$ conjunct of φ. Formally, consider a number $1 \leq i \leq n$ and a subformula $\forall x \exists y\ \chi_i(x, y)$. We require that there exists a 1-type β and an order formula $\theta \in \Theta$ such that $\overline{\alpha}(\theta)(\beta) \neq \vec{0}$ and the logical implication $\alpha(x) \wedge \beta(y) \wedge \theta(x, y) \models \chi_i(x, y)$ holds.
3. Each modulo conjunct has the right number of witnesses. Consider a number $1 \leq j \leq m$ and a subformula $\forall x \exists^{\bowtie_j k_j, l_j} y\ \psi_j(x, y)$. We require that the remainder modulo l_j of the number of witnesses encoded in the full-type satisfies the inequality $\bowtie_j k_j$. Formally,

$$r_{l_j}\left(\sum_{\theta\in\Theta, \beta\in\boldsymbol{\alpha}\,:\,\alpha(x)\wedge\beta(y)\wedge\theta(x,y)\models\psi_j(x,y)} \pi_{j+1}\left(\overline{\alpha}(\theta)(\beta)\right)\right) \bowtie_j k_j.$$

A proof of the following lemma is a straightforward unfolding of Definitions 4 and 5 and the definition of the semantics of $\mathrm{FO}^2_{\mathrm{MOD}}$.

▶ **Lemma 6.** *Assume that a formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}$ in normal form is interpreted over finite words or trees. Then $\mathfrak{M} \models \varphi$ if and only if every $(l_1, l_2, \ldots, l_m)$-full type realized in $\mathfrak{M}$ is φ-consistent.*

Let φ be $\mathrm{FO}^2_{\mathrm{MOD}}$ formula in normal form. For proving the upper bounds in the next sections, we will estimate the size of a model by the following function. We define $\mathfrak{f}(\varphi)$ as the function, which for a given formula returns the total number of $(l_1, l_2, \ldots, l_m)$-full types over the signature of φ. To be precise, $\mathfrak{f}(\varphi)$ is equal to $(2l_1l_2 \ldots l_m)^{|\Theta||\boldsymbol{\alpha}|}$. Note that $\mathfrak{f}(\varphi)$ is doubly exponential in the size of the formula φ.

3 Finite words

In this section we focus our attention on the case of finite words. We give an alternative proof of ExpSpace upper bound for the satisfiability problem of $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$. Originally this result was proved in [15] by a reduction to unary temporal logic with modulo counting operators. Here we give a direct algorithm dedicated to $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$, based on a small model property that we prove. The advantage of the method is that it allows an extension to the case of trees and other extensions (e.g., an incorporation of counting quantifiers of the form $\exists^{\leq k}, \exists^{\geq k}$ is quite obvious).

Small model property. We start by proving that every satisfiable formula in $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$ has a small model. The proof technique is similar to the pumping lemma known from the theory of finite word automata.

▶ **Lemma 7.** *Every normal form $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$ formula satisfiable over finite words has a model $\mathfrak{W}$ of size bounded by $\mathfrak{f}(\varphi)$.*

Proof. Consider a satisfiable formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$ and assume that its model $\mathfrak{W}$ is a word longer than $\mathfrak{f}(\varphi)$. We will show that we can remove some subword from $\mathfrak{W}$ and

Procedure 1: Satisfiability test for $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$

```
Input: Formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$ in normal form.
MaxLength := $\mathfrak{f}(\varphi)$                                  // Maximal length of a model from Lemma 7
CurrentPosition := 0
guess a full type $\overline{\alpha}$ s.t. $\overline{\alpha}(\theta_{\ll})$ and $\overline{\alpha}(\theta_{-1})$ are zero   // Type of the first position
if not is-$\varphi$-consistent($\overline{\alpha}$) then reject                // See Definition 5
while $CurrentPosition < MaxLength$ do
    if both $\overline{\alpha}(\theta_{\gg})$ and $\overline{\alpha}(\theta_{+1})$ are zero then accept   // Type of the last position
    guess a full type $\overline{\beta}$                                 // Type of the successor
    if not is-$\varphi$-consistent($\overline{\beta}$) then reject             // See Definition 5
    if not is-valid-successor($\overline{\beta}, \overline{\alpha}$) then reject
    $\overline{\alpha} := \overline{\beta}$
    CurrentPosition := CurrentPosition + 1
reject
```

obtain a shorter model. By repeating this process, we will finally obtain a model of φ with a required size.

By the pigeonhole principle there exist two positions $u, v \in W$ with equal full-types. Let $\mathfrak{W}'$ be a word obtained from $\mathfrak{W}$ by removing all letters from positions between u and v and collapsing u and v into a single position. Observe that since full types of u and v are equal, for all j the remainders modulo l_j of the total number of removed 1-types on positions between u and v are equal to 0. Note also that due to presence of 0-1 part in the definition of a full type, all unique 1-types realized by the structure survive the surgery. Therefore, all full types in $\mathfrak{W}$ remain unchanged. Since all these full types were φ-consistent in $\mathfrak{W}$, they are also φ-consistent in $\mathfrak{W}'$. As a consequence of Lemma 6, the word $\mathfrak{W}'$ is indeed a model of φ, as expected. ◀

Algorithm. Now we are ready to present an ExpSpace algorithm for solving the satisfiability for $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$ interpreted over finite words. We assume that the input formula φ is in normal form. Since models of φ can have doubly-exponential length, we cannot simply guess a complete intended model. To overcome this difficulty, we guess the model on the fly, letter by letter. For each position of the guessed word we guess its full type and after checking some consistency conditions described in Definition 5, we verify if it can be linked with the previous position. This way we never have to store in memory more than two full types. Since the size of a full type is bounded exponentially in $|\varphi|$, the whole procedure runs in ExpSpace. We accept the input, if the guessing process ends after at most $\mathfrak{f}(\varphi)$ steps.

To avoid presentational clutter in the description of our algorithm we omit the details of two simple tests *is-valid-successor* and *is-φ-consistent*. The former takes two full types, $\overline{\beta}$ and $\overline{\alpha}$, and checks if the position of the type $\overline{\alpha}$ can indeed have a successor of type $\overline{\beta}$. It can be easily done by comparing the total number of 1-types on each relative position stored in both types. The latter test takes one full type $\overline{\alpha}$ and checks if it satisfies the conditions described in Definition 5.

The correctness of the algorithm above is guaranteed by the following lemma.

▶ **Lemma 8.** *Procedure 1 accepts its input φ if and only if φ is satisfiable.*

4 Finite trees

In this section we prove the main result of this paper, which is the following theorem. It is a direct consequence of Theorems 12 and 16 below.

▶ **Theorem 9.** *The satisfiability problem for* $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ *interpreted over finite trees is 2-*ExpTime*-complete.*

4.1 Upper bound

We start by proving the 2-ExpTime upper bound. As in previous section, this is done by showing first a small model property and then an algorithm. The small model property is crucial in the proof of correctness of the algorithm. Actually, having defined the notion of the full type, the technique here is a rather straightforward combination of the technique from previous section and from [4]. The main difficulty here was to come up with the right notion of a full type.

4.1.1 Small model property

We demonstrate the small-model property of the logic $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ by showing that every satisfiable formula φ has a tree model of depth and degree bounded by $\mathfrak{f}(\varphi)$. This is done by first shortening $\downarrow$-paths and then shortening the $\rightarrow$-paths, as in the proof of Lemma 7.

▶ **Theorem 10** (Small model theorem)**.** *Let* φ *be a normal form* $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ *formula. If* φ *is satisfiable then it has a a tree model in which every path has length bounded by* $\mathfrak{f}(\varphi)$ *and every vertex has degree bounded by* $\mathfrak{f}(\varphi)$*.*

Proof. Let $\mathfrak{T}$ be a model of φ. First we show how to shorten $\downarrow$-paths in $\mathfrak{T}$. Assume that there exists a $\downarrow$-path in $\mathfrak{T}$ longer than $\mathfrak{f}(\varphi)$. Thus, by the pigeonhole principle, there are two nodes u and v on this path such that v is a descendant of u and $\mathrm{ftp}^{\mathfrak{T}}_{\varphi}(u) = \mathrm{ftp}^{\mathfrak{T}}_{\varphi}(v)$. Consider the tree $\mathfrak{T}'$, obtained by removing the subtree rooted at u and replacing it by the subtree rooted at v.

Observe that for all j the remainders modulo l_j of the total number of removed vertices is equal to 0. Additionally, all unique full types survive the surgery. Thus, all full types are retained from the original tree, so they are φ-consistent and by Lemma 6 the tree $\mathfrak{T}'$ is a model of φ. By repeating this process we get a tree with all $\downarrow$-paths shorter than $\mathfrak{f}(\varphi)$.

Shortening the $\rightarrow$-paths is done in a similar way. Assume that there exists a node v with branching degree greater than $\mathfrak{f}(\varphi)$. Then there exist two children u, w of v, with equal full types. Let $\mathfrak{T}''$ be a tree obtained by removing all vertices between u and w (excluding u and including w) together with subtrees rooted at them. Again, the remainders modulo l_j of the total number of removed vertices are all equal to 0, and all unique types survive the surgery, so all full types are retained from $\mathfrak{T}$ and thus φ-consistent. Therefore the tree $\mathfrak{T}''$ is a model of φ. We repeat this process until we get a tree with desired branching. ◀

4.1.2 Algorithm

In this section, we design an algorithm to check if a given $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ formula φ interpreted over finite trees is satisfiable. By Lemma 2 we can assume that the input formula φ is given in normal form. Then, by Theorem 10, we can turn our attention to trees with degree and path length bounded by $\mathfrak{f}(\varphi)$. Recall that $\mathfrak{f}(\varphi)$ is doubly-exponential in $|\varphi|$.

Let us describe the core ideas of the algorithm. It works in alternating exponential space. Since AExpSpace = 2-ExpTime, it can be translated to an algorithm working in 2-ExpTime. The algorithm starts by guessing the full type of the root and then it calls a procedure that builds a tree with the given full type of the root.

The procedure called on a vertex v guesses on the fly the children of v, starting with the leftmost child and storing in the memory at most two children at the same time. While guessing a child v', it checks that its type is φ-consistent and that it can be correctly linked to v and to its left sibling (if it exists). Then the procedure is called recursively on v' to build the subtree rooted at v', but the recursive call is done in parallel, exploiting the power of alternation. The process is continued in the same way, until the bottom of the tree is reached or the height of the constructed tree exceeds $\mathfrak{f}(\varphi)$.

There is one more point of the algorithm, namely, we have to make sure that the types of children and their descendants, guessed on the fly, are consistent with the information stored in their parent's full type, in particular with $\theta_{\downarrow}$ and $\theta_{\downarrow\downarrow^+}$ components. To handle the first of them, we can simply compute the union of all $\theta_{=}$ components of children, which can be done during the guessing process. Analogously, to handle the second case, we simply compute the union of all $\theta_{\downarrow}$ and $\theta_{\downarrow\downarrow^+}$ components of all children's full types.

Below we present a pseudocode of the described procedure. Similarly to the case of finite words, we omit the details of obvious methods for checking consistency. For calculating the union of full-types mentioned in the previous paragraph, we employ $\oplus$ operation defined as follows. For given $(l_1, l_2, \ldots, l_m)$-full types $\overline{\alpha}$ and $\overline{\beta}$, the result of $\overline{\alpha} \oplus \overline{\beta}$ is a $(l_1, l_2, \ldots, l_m)$-full type $\overline{\gamma}$ such that for all order formulas $\theta \in \Theta$ and all 1-types $\alpha \in \boldsymbol{\alpha}$, the following condition holds:

$$\overline{\gamma}(\theta)(\alpha) = \big(\max\big(\pi_0(\overline{\alpha}(\theta)(\alpha)), \pi_0(\overline{\beta}(\theta)(\alpha))\big), R_1, R_2, \ldots, R_m\big),$$

where $R_i = r_{l_i}(\pi_i(\overline{\alpha}(\theta)(\alpha)) + \pi_i(\overline{\beta}(\theta)(\alpha)))$.

The following lemma guarantees the correctness of the algorithm and leads directly to the main result of this section.

▶ **Lemma 11.** *Procedure 3 accepts its input formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ if and only if φ is satisfiable over finite trees.*

▶ **Theorem 12.** *The satisfiability problem for $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ interpreted over finite trees is in 2-ExpTime.*

4.2 Lower bound

In this section we prove that the satisfiability of $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$ is 2-ExpTime-hard. We exploit here the fact that 2-ExpTime =AExpSpace and provide a generic reduction from AExpSpace. This is done in two steps. First we translate computations of alternating Turing machines to winning strategies in (our version of) tiling games. These strategies are then encoded as trees and their existence is translated to the satisfiability problem in $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$.

Alternating Turing machines. An alternating Turing machine is Turing machine whose set of states contains *existential* states and *universal* states. A input word is *accepted* by such a machine if the initial configuration *leads to acceptance*, where leading to acceptance is defined recursively as follows: accepting configurations (i.e., configurations containing

Procedure 2: Building a subtree rooted at given node

```
   Input: Formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ in normal form,
   full type $\overline{\alpha}$ of a starting node, and current level $\mathrm{Lvl} \in \mathbb{N}$.
1  if not is-$\varphi$-consistent($\overline{\alpha}$) then reject                    // See Definition 5
2  if $\mathrm{Lvl} \geq \mathfrak{f}(\varphi)$ then reject                              // Path too long
3  if $\overline{\alpha}(\theta_{\downarrow})$ is zero then accept                      // Last node on the path
4  Guess the degree $\mathrm{Deg} \in [1, \mathfrak{f}(\varphi)]$ of a node
5  Guess the full type $\overline{\beta}$ of the leftmost child and check if its a valid leftmost son of $\overline{\alpha}$
6  $O_{\theta_{\downarrow}} := \overline{\beta}(\theta_{=})$                               // Types of children guessed so far
7  $O_{\theta_{\downarrow\downarrow^+}} := \overline{\beta}(\theta_{\downarrow}) \oplus \overline{\beta}(\theta_{\downarrow\downarrow^+})$    // Types of descendants guessed so far
8  while $Deg > 1$ do
9      Run in parallel Procedure 2 on $(\varphi, \overline{\beta}, \mathrm{Lvl} + 1)$         // Alternation here
10     Guess a full type $\overline{\gamma}$ of the right brother of $\overline{\beta}$ and check consistency with $\overline{\alpha}$
11     $O_{\theta_{\downarrow}} := O_{\theta_{\downarrow}} \oplus \overline{\gamma}(\theta_{=})$, $O_{\theta_{\downarrow\downarrow^+}} := O_{\theta_{\downarrow\downarrow^+}} \oplus \overline{\gamma}(\theta_{\downarrow}) \oplus \overline{\gamma}(\theta_{\downarrow\downarrow^+})$   // Updating obligations
12     $\overline{\beta} := \overline{\gamma}$, $\mathrm{Deg} := \mathrm{Deg} - 1$
13 Run in parallel Procedure 2 on $(\varphi, \overline{\beta}, \mathrm{Lvl} + 1)$             // Last child
14 if $\overline{\beta}(\theta_{\rightarrow})$ is not zero then reject                    // Not valid last node on $\rightarrow$-path.
15 if $\overline{\alpha}(\theta_{\downarrow}) = O_{\theta_{\downarrow}}$ and $\overline{\alpha}(\theta_{\downarrow\downarrow^+}) = O_{\theta_{\downarrow\downarrow^+}}$ then accept else reject
```

Procedure 3: Satisfiability test for $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$

```
   Input: Formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ in normal form.
1  guess a full type $\overline{\alpha}$ s.t. $\overline{\alpha}(\theta_{\uparrow})$ is zero       // Type of the root
2  Run Procedure 2 on $(\varphi, \overline{\alpha}, 1)$
```

accepting state) lead to acceptance; an existential configuration leads to acceptance if there exists its successor configuration that leads to acceptance; a universal configuration leads to acceptance if all its successor configurations lead to acceptance. More details can be found in [2].

4.2.1 Tilling Games

Corridor tiling games, aka rectangle tiling games [8] provide a well-known technique for proving lower bounds in space complexity. Here we develop our own version of these games that is able to encode alternating Turing machines from previous section and to be encoded in $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$.

By a tiling game we understand a tuple of the form $\langle C, T_0, T_1, n, \langle t_0, \ldots, t_n \rangle, \Box, L \rangle$, where C is a finite set of *colors*; $T_0, T_1 \subseteq C^4$ are two sets of *tiles* (these two sets are not meant to be disjoint); n is a natural number; $\langle t_0, \ldots, t_n \rangle$ is an *initial* sequence of $n+1$ tiles; $\Box \in C$ is a special color called *white*; $L \subseteq T_0 \cup T_1$ is a set of tiles allowed in the *last* row.

We think of a tile $\langle a, b, c, d \rangle \in C^4$ as of a square consisting of four smaller squares, colored respectively with colors a, b, c and d (see Figure 1). In the following we will require that adjacent tiles have matching colors, both horizontally and vertically. Formally, we define the horizontal adjacency relation $H = \{\langle \langle a, b, c, d \rangle, \langle b, e, d, f \rangle \rangle \mid a, b, c, d, e, f, \in C\}$ and the vertical adjacency relation $V = \{\langle \langle a, b, c, d \rangle, \langle c, d, e, f \rangle \rangle \mid a, b, c, d, e, f, \in C\}$. We define a *correctly tiled corridor* to be a rectangle of size $k \times 2^n$ for some $k \in \mathbb{N}$ filled with tiles, with all horizontally adjacent tiles in H and all vertically adjacent tiles in V, with first row starting

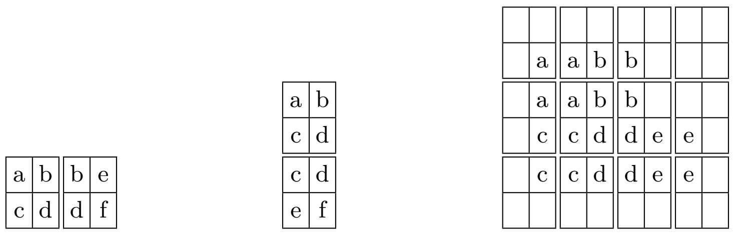

Figure 1 Horizontally adjacent tiles, vertically adjacent tiles, and a correct tiling

with tiles $t_0, \ldots, t_n$ and padded out with white tiles, with last row built only from tiles in L, and with all edges being white. Figure 1 shows an example of correctly tiled corridor for $n = 2$ and initial tiles $\langle \square, \square, \square, a\rangle, \langle \square, \square, a, b\rangle, \langle \square, \square, b, \square\rangle$.

Consider the following game. There are two players, *Prover* (also called the *existential* player) and *Spoiler* (also called the *universal* player). The task of Prover is to construct a correct tiling; the task of Spoiler is to prevent Prover from doing this. At the beginning they are given the initial row $t_0, \ldots, t_n, (\square^4)^{2^n-n-1}$. In each move the players alternately choose one of the two sets T_0 or T_1 and build one row consisting of 2^n tiles from the chosen set. The first move is performed by Prover. Prover wins if after a finite number of moves there is a correctly tiled corridor, otherwise Spoiler wins. There are two possibilities for Spoiler to win: either Prover cannot make a move while the last constructed row is not in L^* or the game lasts forever.

We say that a tiling game $\langle \_, T_0, T_1, \_, \_, \_, \_\rangle$ is *well-formed* if for any play of the game and for any partial (i.e., non-complete) tiling constructed during this play, exactly one new row can be correctly constructed from tiles in T_0 and exactly one from tiles in T_1. In other words, every possible move in any play of the game is fully determined by the choice of the set of tiles, T_0 or T_1.

4.2.2 From Alternating Machines to Tiling Games

The first step of the lower-bound proof for $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$ is a reduction from alternating machines to tiling games. Actually, we have defined our version of tiling games in such a way that the proof of the following theorem becomes a routine exercise. The details can be found in the full version.

▶ **Theorem 13.** *For all alternating Turing machines M working in exponential space and all input words w there exists a well-formed tiling game of size polynomial in the sizes of M and w such that Prover has a winning strategy in the game if and only if w is accepted by M.*

The theorem above directly leads to the lower bound on the complexity of tiling games.

▶ **Corollary 14.** *The problem whether Prover has a winning strategy in a tiling game is 2-*EXPTIME*-hard.*

4.2.3 From Tilling Games to $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$

Here we show the second step of the lower-bound proof for $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$, which is a reduction from tiling games to $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$. We are going to encode strategies for the existential player as trees. Every complete path in such an encoding corresponds to a correct tiling for G. The nodes on such a path, read from the root to the leaf, correspond to tiles in the tiling, read row by row from left to right. Intuitively, most nodes have just one child corresponding to the tile placed directly to the right. Nodes corresponding to tiles in the last column should have one or two children, depending on whether it is the existential or universal player's turn. Formally the situation is a bit more complicated, because we are not able to prevent the tree from having additional branches with no meaning (for example, a node may have several children, each of which encodes the same right neighbor).

For a given number n we will be using unary predicates $B_0, \dots, B_{n-1}$ for counting modulo 2^n; the predicate B_i will be responsible of the i-th bit of the corresponding number; B_0 is the least significant bit. This is a standard construction that can be found e.g. in [13] or [4], so we do not present the details here. In the following we will be using several macros that expand in an obvious way to formulas of length polynomial in n over predicates $B_0, \dots, B_{n-1}$. Some of them are listed below.

$Nr(x) = 0$	the number encoded in the node x is 0
$Nr(x) = 2^n - 1$	the number encoded in the node x is $2^n - 1$
$Nr(x) = Nr(y) + 1$	the number in x is the successor of the number in y
$Nr(x) > Nr(y)$	the number in x is greater than the number in y
$\dots$	$\dots$

For example, the macro $Nr(x) > Nr(y)$ expands to

$$\bigvee_{i=0}^{n} \Big(B_i(x) \land \neg B_i(y) \land \bigwedge_{j=i+1}^{n} (B_j(x) \Leftrightarrow B_j(y)) \Big)$$

▶ **Theorem 15.** *For all well-formed tiling games G there exists a formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$ of size polynomial in the size of G such that the existential player has a winning strategy in G if and only if φ is satisfiable over finite trees.*

Sketch of proof. Let $G = \langle C, T_0, T_1, n, \langle t_0, \dots, t_n \rangle, \Box, L \rangle$. We are going to define the formula φ as a conjunction of several smaller formulas responsible for different aspects of the encoding. Most of these aspects are routine; the most interesting is adjacency.

Numbering of nodes. We start by numbering nodes on paths in the underlying tree. These numbers encode the column numbers of the corresponding tiles. The following formulas express that the root is numbered 0, the number of any other node is the number of its father plus one modulo 2^n, and that all rows are complete (i.e., they have 2^n tiles).

$$\forall x \big(\neg (\exists y \; y{\downarrow}x) \Rightarrow Nr(x) = 0 \big)$$
$$\forall x \big(Nr(x) \neq 2^n - 1 \Rightarrow (\exists y \; x{\downarrow}y) \land \forall y \; (x{\downarrow}y \Rightarrow Nr(y) = Nr(x) + 1) \big)$$
$$\forall x \big(Nr(x) = 2^n - 1 \Rightarrow \forall y \; (x{\downarrow}y \Rightarrow Nr(y) = 0) \big)$$

Tiles and colors. Let $t_0, \dots, t_m$ be an enumeration of all tiles occurring in the game, that is in $T_0 \cup T_1 \cup \{t_0, \dots, t_n, \langle \Box, \Box, \Box, \Box \rangle\}$. The predicates $tile_1, \dots, tile_m$ correspond to these tiles. For each color $c \in C$ we introduce four predicates $\pi_1^c, \pi_2^c, \pi_3^c, \pi_4^c$ that will be used to encode the four colors of a tile. The formulas below express that each node in the underlying tree corresponds to precisely one tile and is colored with the four colors of the tile. We assume here that $t_i = \langle c_i^1, c_i^2, c_i^3, c_i^4 \rangle$. We also introduce predicates $setT_0$, $setT_1$ and $setL$ corresponding to the sets T_0, T_1 and L, respectively.

$$\forall x\Big(\bigvee_{i=0}^{m}\big(tile_i(x)\wedge\bigwedge_{j\neq i}\neg tile_j(x)\big)\Big)$$

$$\forall x\bigwedge_{i=1}^{4}\Big(\bigvee_{c\in C}^{m}\big(\pi_i^c(x)\wedge\bigwedge_{c'\neq c}\neg\pi_i^{c'}(x)\big)\Big)$$

$$\bigwedge_{i=0}^{m}\Big(\forall x\big(tile_i(x)\Leftrightarrow\pi_1^{c_i^1}(x)\wedge\pi_2^{c_i^2}(x)\wedge\pi_3^{c_i^3}(x)\wedge\pi_4^{c_i^4}(x)\big)\Big)$$

$$\Big((\bigvee_{t_i\in T_0} tile_i(x))\Leftrightarrow setT_0(x)\Big)\wedge\Big((\bigvee_{t_i\in T_1} tile_i(x))\Leftrightarrow setT_1(x)\Big)\wedge\Big((\bigvee_{t_i\in L} tile_i(x))\leftrightarrow setL(x)\Big)$$

First and last row. The predicates *First* and *Last* are used to distinguish the first and the last row of a correct tiling. A node x corresponds to a tile in the first row if there is no other tile in the same column in previous rows. The last row is described dually. The first row is built from tiles $t_0,\dots,t_n$ and padded out with white tiles, the last row is built only from tiles in L. The formulas expressing these properties are quite obvious and we omit them here, but they can be found in [2].

Existential and universal rows. Each element in each row is marked with predicate E or A depending on which player's turn it is. Each row is marked the same (each element has the same marking as its left neighbor, if it exists). The first row is existential and then the marking alternates between existential and universal.

Universal rows have two successors. Each non-first row is marked with predicate $move_0$ or $move_1$ depending on the set of tiles (T_0 or T_1) from which it is built. Universal non-last rows have two successors, marked respectively with $move_0$ and $move_1$.

Horizontal adjacency. This is simple. We first establish the white frame on the first and last tile in each row and then simply say that for each non-first tile in any row the left edge of the tile matches the right edge of the preceding tile.

$$\forall x\ \big(Nr(x)=0\Rightarrow\pi_1^{\Box}(x)\wedge\pi_3^{\Box}(x)\big)$$

$$\forall x\ \big(Nr(x)=2^n-1\Rightarrow\pi_2^{\Box}(x)\wedge\pi_4^{\Box}(x)\big)$$

$$\bigwedge_{c\in C}\Big(\forall x\big((Nr(x)\neq 0)\wedge\pi_1^c(x)\Rightarrow\exists y\ (y{\downarrow}x\wedge\pi_2^c(y))\big)\Big)$$

$$\bigwedge_{c\in C}\Big(\forall x\big((Nr(x)\neq 0)\wedge\pi_3^c(x)\Rightarrow\exists y\ (y{\downarrow}x\wedge\pi_4^c(y))\big)\Big)$$

Vertical adjacency. This is the tricky part of the encoding. The difficulty comes from the fact that we cannot number rows of the tiling and we have no means to say that two tiles occur in consecutive rows. Using predicates $B_0,\dots,B_{n-1}$ we can number the 2^n columns, but the number of rows may be much higher and we cannot afford having enough predicates for numbering them. Therefore, we can say that two tiles occur in the same column (or in consecutive columns), but we cannot do the same with rows. On the other hand, when we add a new tile to a row, we have to make sure that the upper edge of the new tile matches the lower edge of the tile directly above, so we have to read the colors of the tile directly above. For non-white colors this can be done by observing that each occurrence of a color in an upper edge of a row must be matched with another occurrence of the same color in a lower edge of the previous row in the same column, therefore the number of occurrences of each color in each column should be even. Hence the color directly above the upper edge of the current row is the only non-white color that occurs an odd number of times in the current column in preceding rows. In the case of white color we have to take into consideration the upper white edge of the constructed tiling, so the color directly above is white if it occurs an even number of times in the current column in preceding rows.

$$\forall x\big(\pi_1^{\Box}(x) \Rightarrow \exists^{=0,2} y\ \ y{\downarrow^+}x \wedge Nr(y){=}Nr(x) \wedge (\pi_1^{\Box}(y) \vee \pi_3^{\Box}(y))\big)$$
$$\forall x\big(\pi_2^{\Box}(x) \Rightarrow \exists^{=0,2} y\ \ y{\downarrow^+}x \wedge Nr(y){=}Nr(x) \wedge (\pi_2^{\Box}(y) \vee \pi_4^{\Box}(y))\big)$$
$$\bigwedge_{c \in C \setminus \{\Box\}} \Big(\forall x\big(\pi_1^{c}(x) \Rightarrow \exists^{=1,2} y\ y{\downarrow^+}x \wedge Nr(y){=}Nr(x) \wedge (\pi_1^{c}(y) \vee \pi_3^{c}(y))\big)\Big)$$
$$\bigwedge_{c \in C \setminus \{\Box\}} \Big(\forall x\big(\pi_2^{c}(x) \Rightarrow \exists^{=1,2} y\ y{\downarrow^+}x \wedge Nr(y){=}Nr(x) \wedge (\pi_2^{c}(y) \vee \pi_4^{c}(y))\big)\Big)$$

Correctness of the constructed formula. Let φ be the conjunction of all formulas mentioned above. It is not difficult to see that the size of φ is polynomial in the size of G: the longest part of φ is the encoding of tiles and colors, which is proportional in length to the sum of squared numbers of tiles and colors.

Assume that φ has a finite model $\mathfrak{M}$. The formula *Tiles and colors* guarantees that each node in $\mathfrak{M}$ directly encodes precisely one tile. The formula *Numbering of nodes* guarantees that nodes on each root-to-leaf path are correctly numbered modulo 2^n, with the root numbered 0 and the leaf numbered $2^n - 1$. Thus each segment of length 2^n of such a path, consisting of nodes numbered from 0 to $2^n - 1$, corresponds to one row of tiles. The *Horizontal adjacency* formula guarantees that (horizontally) adjacent tiles in such a row have matching colors. Similarly, *Vertical adjacency* formula guarantees that tiles occurring on the same position in two consecutive rows (that is, vertically adjacent tiles) have matching colors. The *First and last row* formula guarantees that the first row is initial and the last row is built from tiles in L. Therefore each complete path in $\mathfrak{M}$ encodes a correctly tiled corridor. By the *Existential and universal rows* formula all rows encoded in any such path are alternately marked as existential and universal, starting with an existential one. Finally, the *Universal rows have two successors* formula guarantees that each encoding of a universal row in $\mathfrak{M}$ is followed by encodings of two existential rows, one build from tiles in T_0 and one from tiles in T_1. Thus (here we use the assumption that the game is well-formed) $\mathfrak{M}$ covers both possible moves of the universal player. Now the strategy of the existential player is to follow the path in $\mathfrak{M}$ corresponding to the partial tiling as it is constructed during the game. She starts in the root of $\mathfrak{M}$. Then, every time when it is her turn, the existential player reads from $\mathfrak{M}$ any successor row of her current position and replies with this row, moving down the tree to the position of the successor row. Every time when it is the universal player's turn, his both possible moves are encoded as successor rows of the current position of the existential player, so she can always follow the branch chosen by the universal player. Since $\mathfrak{M}$ is finite, each play stops after finitely many rounds with the constructed tiling corresponding to a path in $\mathfrak{M}$. Therefore in each play we obtain a correctly tiled corridor and the existential player wins.

For the other direction, assume that the existential player has a winning strategy. We construct a model $\mathfrak{M}$ of φ inductively, level by level, as follows. We start with 2^n nodes, number them from 0 to $2^n - 1$ using predicates $B_0, \dots, B_{n-1}$ and connect consecutive nodes with predicate $\downarrow$. Then we label nodes numbered 0 to n with predicates $\mathit{tile}_0, \dots, \mathit{tile}_n$, respectively, and nodes numbered $n+1$ to $2^n - 1$ with $\langle \square, \square, \square, \square \rangle$. Then (and later, always after adding new labels of the form tile_i) we add appropriate labels $\pi^c_j, \mathit{setT}_0, \mathit{setT}_1$ and setL to make the formula *Tiles and colors* true. Similarly, we add labels $\downarrow^+$ to all pairs of nodes that are connected by the transitive closure of $\downarrow$. We also label all these nodes with predicates *First* and E. This way we obtain the encoding of the initial row in the game. Next we inductively, level by level, construct the remaining parts of $\mathfrak{M}$.

Assume that $\mathfrak{M}$ is constructed up to some level k, with all leaves labeled E, and that each path constructed so far encodes a partial tiling in some play after k rounds, with existential player's turn. We extend $\mathfrak{M}$ leaf by leaf. Let us consider one such leaf ℓ, it encodes the last tile in some row. If all tiles in this row are in the set L, we label all nodes in this row with predicate *Last* and finish the construction. Otherwise let r be the next row given by the winning strategy of the existential player in the partial play encoded by the path from the root to ℓ. We take fresh 2^n nodes and extend the path from root to ℓ with the encoding of a row r, as above, but this time labeling all new nodes with predicate A indicating the universal player's move. We also label all new nodes with predicate move_0 or move_1, depending on whether r is constructed from tiles in T_0 or T_1, respectively. Again, if the row is final, it is marked with predicate *Last*, otherwise we take twice 2^n fresh nodes and connect to the current leaf two segments of length 2^n that encode two possible moves of the universal player. We mark each node in both segments with predicate E and thus finish the construction of level $k+1$ at node ℓ. The construction is repeated for all leafs at level k. Since during the construction we follow the winning strategy, no encoded play can last forever and the construction ends after finitely many iterations. By inspection of all conjuncts one can check that the constructed tree is a model of φ. ◀

As a corollary of the theorem above and Corollary 14 we get the main theorem of this section.

▶ **Theorem 16.** *The satisfiability problem for* $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$ *interpreted over finite trees is 2-*EXPTIME*-hard.*

5 Conclusions and future work

We have shown that the satisfiability problem for two-variable logic extended with modulo counting quantifiers and interpreted over finite trees is 2-EXPTIME-complete. The upper bound is based on the small-model property of the logic; for the lower bound we have developed a version of tiling games for AEXPSPACE computations.

There are several possible directions for future work. One of them is studying restrictions or extensions of the logic presented here. Natural candidates for restrictions include *guarded fragment* of the logic or *unary alphabet restriction* as in [4]; natural extensions include arbitrary uninterpreted binary symbols as in [3]. Another possibility is investigation of the (finite) satisfiability problem for $\mathrm{FO}^2_{\mathrm{MOD}}$ on arbitrary structures – we even do not know whether this problem is decidable. Yet another direction is to study the expressive power of the logic and to find an expressively equivalent extension of CTL.

References

1 Augustin Baziramwabo, Pierre McKenzie, and Denis Thérien. Modular temporal logic. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 344–351. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782629.

2 Bartosz Bednarczyk and Witold Charatonik. Modulo counting on words and trees. *CoRR*, http://arxiv.org/abs/1710.05582, 2017. URL: http://arxiv.org/abs/1710.05582.

3 Bartosz Bednarczyk, Witold Charatonik, and Emanuel Kieronski. Extending two-variable logic on trees. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 11:1–11:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CSL.2017.11.

4 Saguy Benaim, Michael Benedikt, Witold Charatonik, Emanuel Kieroński, Rastislav Lenhardt, Filip Mazowiecki, and James Worrell. Complexity of two-variable logic on finite trees. *ACM Transactions on Computational Logic*, 17(4):32:1–32:38, 2017. Extended abstract in ICALP 2013. URL: http://dl.acm.org/citation.cfm?id=2996796.

5 Michael Benedikt and Luc Segoufin. Regular tree languages definable in FO and in fo$_{mod}$. *ACM Trans. Comput. Log.*, 11(1):4:1–4:32, 2009. doi:10.1145/1614431.1614435.

6 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICDT.2017.8.

7 Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and a linear order. *Logical Methods in Computer Science*, 12(2), 2016. doi:10.2168/LMCS-12(2:8)2016.

8 Bogdan S. Chlebus. Domino-tiling games. *J. Comput. Syst. Sci.*, 32(3):374–392, 1986. doi:10.1016/0022-0000(86)90036-X.

9 Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002. doi:10.1006/inco.2001.2953.

10 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 306–317. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614957.

11 Frederik Harwath and Nicole Schweikardt. On the locality of arb-invariant first-order formulas with modulo counting quantifiers. *Logical Methods in Computer Science*, 12(4), 2016. doi:10.2168/LMCS-12(4:8)2016.

12 Lucas Heimberg, Dietrich Kuske, and Nicole Schweikardt. Hanf normal form for first-order logic with unary counting quantifiers. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 277–286. ACM, 2016. doi:10.1145/2933575.2934571.

13 Emanuel Kieronski. On the complexity of the two-variable guarded fragment with transitive guards. *Inf. Comput.*, 204(11):1663–1703, 2006. doi:10.1016/j.ic.2006.08.001.

14 Andreas Krebs and A. V. Sreejith. Non-definability of languages by generalized first-order formulas over (n, +). In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 451–460. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.55.

15 Kamal Lodaya and A. V. Sreejith. Two-variable first order logic with counting quantifiers: Complexity results. In Émilie Charlier, Julien Leroy, and Michel Rigo, editors, *Developments in Language Theory - 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 260–271. Springer, 2017. doi:10.1007/978-3-319-62809-7_19.

16 Maarten Marx and Maarten de Rijke. Semantic characterization of navigational XPath. In *First Twente Data Management Workshop (TDM 2004) on XML Databases and Information Retrieval*, pages 73–79, 2004.

17 Juha Nurmonen. Counting modulo quantifiers on finite structures. *Inf. Comput.*, 160(1-2):62–87, 2000. doi:10.1006/inco.1999.2842.

18 Andreas Potthoff. Modulo-counting quantifiers over finite trees. *Theor. Comput. Sci.*, 126(1):97–112, 1994. doi:10.1016/0304-3975(94)90270-4.

19 Amitabha Roy and Howard Straubing. Definability of languages by generalized first-order formulas over n^+. *SIAM J. Comput.*, 37(2):502–521, 2007. doi:10.1137/060658035.

20 Dana Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:477, 1962.

21 A. V. Sreejith. Expressive completeness for LTL with modulo counting and group quantifiers. *Electr. Notes Theor. Comput. Sci.*, 278:201–214, 2011. doi:10.1016/j.entcs.2011.10.016.

22 Howard Straubing and Denis Thérien. Modular quantifiers. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas].*, volume 2 of *Texts in Logic and Games*, pages 613–628. Amsterdam University Press, 2008.

23 Howard Straubing, Denis Thérien, and Wolfgang Thomas. Regular languages defined with generalized quanifiers. *Inf. Comput.*, 118(2):289–301, 1995. doi:10.1006/inco.1995.1067.