Two-Variable Logic on 2-Dimensional Structures

Amaldev Manuel[1] **and Thomas Zeume**[2]

1 **LIAFA, Université Paris Diderot**
`amal@liafa.univ-paris-diderot.fr`
2 **TU Dortmund University**
`thomas.zeume@cs.tu-dortmund.de`

Abstract

This paper continues the study of the two-variable fragment of first-order logic (FO^2) over two-dimensional structures, more precisely structures with two orders, their induced successor relations and arbitrarily many unary relations. Our main focus is on ordered data words which are finite sequences from the set $\Sigma \times \mathcal{D}$ where Σ is a finite alphabet and $\mathcal{D}$ is an ordered domain. These are naturally represented as labelled finite sets with a linear order $\leq_l$ and a total preorder $\leq_p$.

We introduce ordered data automata, an automaton model for ordered data words. An ordered data automaton is a composition of a finite state transducer and a finite state automaton over the product Boolean algebra of finite and cofinite subsets of $\mathbb{N}$. We show that ordered data automata are equivalent to the closure of $\mathrm{FO}^2(+1_l, \leq_p, +1_p)$ under existential quantification of unary relations. Using this automaton model we prove that the finite satisfiability problem for this logic is decidable on structures where the $\leq_p$-equivalence classes are of bounded size. As a corollary, we obtain that finite satisfiability of FO^2 is decidable (and it is equivalent to the reachability problem of vector addition systems) on structures with two linear order successors and a linear order corresponding to one of the successors. Further we prove undecidability of FO^2 on several other two-dimensional structures.

1998 ACM Subject Classification F.4.1

Keywords and phrases FO^2, Data words, Satisfiability, Decidability, Automata

1 Introduction

The undecidability of the satisfiability and finite satisfiability problem for first-order logic [7, 33, 32] lead to a quest for decidable yet expressive fragments (see for example [4, 16]).

Here we continue the study of the two-variable fragment of first order logic (two-variable logic or FO^2 for short). This fragment is known to be reasonably expressive and its satisfiability and finite satisfiability problems are decidable [26], in fact they are complete for NExpTime [12]. Unfortunately many important properties as for example transitivity cannot be expressed in two-variable logic. This shortcoming led to an examination of extensions of two-variable logic by special relation symbols that are interpreted as equivalence relations or orders [27, 2, 20, 21, 19, 29, 31].

In this paper we are interested in extensions of two-variable logics by two orders and their induced successors. This can be seen as two-variable logic on 2-dimensional structures. We restrict our attention to linear orders and preorders[1]. This setting yields some interesting applications.

Data words, introduced in [5], extend usual words by assigning data values to every position. Applications of data words arise for example in verification, where they can be used

[1] Informally, a *preorder* is an equivalence relation whose equivalence classes are ordered by a linear order.

for modeling runs of infinite state systems, and in database theory, where XML trees can be modeled by data trees. Data words with a linearly ordered data domain can be seen as finite structures with a linear order on the positions and a preorder on the positions induced by the linear order of the data domain. Those relations, as well as their induced successor relations, can then be referred to by two-variable logic on data words [2].

Two other logics closely related to two-dimensional two-variable logic are *compass logic* and *interval temporal logic*. In compass logic two-dimensional temporal operators allow for moving north, south, east and west along a grid [34]. In interval temporal logic operators like 'after', 'during' and 'begins' allow for moving along intervals [15]. The connection of intervals to the two-dimensional setting becomes clear when one interprets an interval $[a, b]$ as point (a, b). In [28] decidability results for two-variable logic in the two-dimensional setting have been transferred to those two logics.

Those applications motivate working towards a thorough understanding of 2-dimensional two-variable logic in general, and the decidability frontier for the finite satisfiability problem in this setting in particular. Next we discuss the state-of-the-art in this area and how our results fit in. All those results are summarized in Figure 3.

The frontier for decidability of the finite satisfiability problem for the extension of two-variable logic by two linear order relations and their induced successor relations is well-understood. It is undecidable when all those relations can be accessed by the logic. It is decidable when only the two successor relations can be accessed [24]. This paper contains a gap (the reduction to Presburger automata is wrong) which can, however, be fixed using the same technique. In [11] an optimal decision procedure is given that uses a different approach; and more recently the result has been generalized to two-variable logic with counting on structures with two trees using yet another approach [6]. When two linear orders and one of their successors can be accessed the problem is decidable as well [28]. We prove that the remaining open case of two successors and one corresponding linear order is decidable.

The addition of two preorders to two-variable logic yields an undecidable finite satisfiability problem [28]. We prove that also the other cases, that is (1) adding two preorder successor relations and (2) adding one preorder relation and one (possibly non-corresponding) preorder successor yield an undecidable finite satisfiability problems.

For the extension of two-variable logic with one linear order, one preorder and their induced successors the picture is not that clear. However, many of the results from above translate immediately, because in two-variable logic one can express that a preorder relation is a linear order. Besides those inherited results the following is known for the finite satisfiability problem. If the access is restricted to one linear order as well as a preorder and its successor, then it is decidable in ExpSpace [28]. Access to a linear order with its successor and either preorder or preorder successor yields undecidability. The former is proved in [3], the latter is an easy adaption. The only remaining open case is when one linear successor, one preorder successor and (possibly) the corresponding preorder can be accessed. We attack this case, and show that when the preorder is restricted to have equivalence classes of bounded size, then the finite satisfiability problem is decidable. The general case was shown to be undecidable after the submission of this work, see Section 7.

Contributions. Besides the above mentioned results, we contribute as follows:

- We introduce *ordered data automata*, an automaton model for structures with one successor relation (of an underlying linear order) and a preorder and its accompanying successor relation. This model is an adaption of data automata, introduced in [3], to data words with an ordered data domain.
- Ordered data automata are shown to be equivalent to the existential two-variable fragment

of monadic second order logic (EMSO^2) over such structures.

- We prove that the emptiness problem for this automaton model is decidable, when the equivalence classes of the preorder contain a bounded number of elements. The decidability of the finite satisfiability problem of two-variable logic over structures with two linear successor relations and one of their corresponding orders is a corollary.

Organization. After some basic definitions in Section 2, we introduce ordered data automata in Section 3 and prove that they are expressively equivalent to $\text{EMSO}^2(+1_l, +1_p, \leq_p)$ in Section 4. Section 5 is devoted to proving decidability of the emptiness problem for ordered data automata when the equivalence classes of $\leq_p$ are bounded. In Section 6 lower bounds for several variants are proved. We conclude with a discussion of recent developments as well as open problems in Section 7. Due to the space limit, most proofs will only be available in the full version of the paper.

Acknowledgements. We thank Thomas Schwentick for introducing us to two-variable logic and for many helpful discussions. The second author acknowledges the financial support by the German DFG under grant SCHW 678/6-1.

2 Preliminaries

We denote the set $\{0, 1, \ldots\}$ of natural numbers by $\mathbb{N}$ and $\{1, \ldots, n\}$, for $n \in \mathbb{N}$ by $[n]$.

A binary relation $\leq_p$ over a finite set A is a *preorder*[2] if it is reflexive, transitive and total, that is, if for all elements u,v and w from A (i) $u \leq_p u$ (ii) $u \leq_p v$ and $v \leq_p w$ implies $u \leq_p w$ and (iii) $u \leq_p v$ or $v \leq_p u$ holds. A *linear order* $\leq_l$ on A is an antisymmetric total preorder, that is, if $u \leq_l v$ and $v \leq_l u$ then $u = v$. Thus, the essential difference between a total preorder and a linear order is that the former allows for two distinct elements u and v that both $u \leq_p v$ and $v \leq_p u$ hold. We call two such elements *equivalent with respect to* $\leq_p$ and denote this by $u \sim_p v$. Hence, a total preorder can be seen as an equivalence relation $\sim_p$ whose equivalence classes are linearly ordered by a linear order. Clearly, every linear order is a total preorder with equivalence classes of size one. We write $u <_l v$ if $u \leq_l v$ but not $v \leq_l u$, analogously for a preorder order $\leq_p$. Further, if C and C' are the equivalence classes of u and v, respectively, then we write $C \leq_p C'$ if $u \leq_p v$.

For a linear order $\leq_l$ an induced *successor relation* $+1_l$ can be defined in the usual way, namely by letting $+1_l(u, v)$ if and only if $u <_l v$ and there is no w with $u <_l w <_l v$. Similarly a preorder $\leq_p$ induces a successor relation $+1_p$ based on the linear order on its equivalence classes, i.e. $+1_p(u, v)$ if and only if $u <_p v$ and there is no w with $u <_p w <_p v$. Thus an element can have several successor elements in $+1_p$.

Two elements u and v are called $\leq_p$*-close* (alternatively $+1_p$*-close*), if either $+1_p(u, v)$ or $u \sim_p v$ or $+1_p(v, u)$. They are called $\leq_p$*-adjacent* (alternatively $+1_p$*-adjacent*) if they are $\leq_p$-close but $u \sim_p v$ does not hold. Analogously for $+1_l$-close, $\leq_l$-close, $+1_l$-adjacent and $\leq_l$-adjacent. The elements u and v are far away with respect to $\leq_p$ if they are not $\leq_p$-close etc. By $u \ll_p v$ we denote that u and v are $\leq_p$-far away and $u \leq_p v$.

In this paper, linear orders and their induced successor relations will be denoted by $\leq_l, \leq_{l_1}, \leq_{l_2}, \ldots$ and $+1_l, +1_{l_1}, +1_{l_2}, \ldots$. Analogously preorders and their induced successor relations will be denoted by $\leq_p, \leq_{p_1}, \leq_{p_2}, \ldots$ and $+1_p, +1_{p_1}, +1_{p_2}, \ldots$.

Ordered Structures, Words and Preorder Words. In this article, an *ordered structure* is a finite structure with non-empty universe and some linear orders, some total

[2] In this paper all preorders are total.

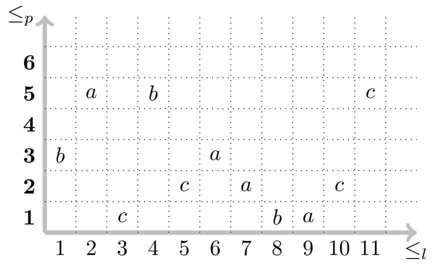


Figure 1 The ordered structure representing the ordered data word $(b,\mathbf{3})(a,\mathbf{5})(c,\mathbf{1})(b,\mathbf{5})(c,\mathbf{2})(a,\mathbf{3})(a,\mathbf{2})(b,\mathbf{1})(a,\mathbf{1})(c,\mathbf{2})(c,\mathbf{5})$. The classes are $\{3,8,9\} \leq_p \{5,7,10\} \leq_p \{1,6\} \leq_p \{2,4,11\}$, the string projection is *bacbcaabacca*, and the preorder projection is $(1,1,1)(1,0,2)(1,1,0)(1,1,1)$ where, e.g., $(1,0,2)$ indicates that in class $\{5,7,10\}$ there is one a-labeled element, no b-labeled element and two c-labeled elements.

preorders, some successor relations and some unary relations. An O-structure is a structure with some unary relations and some binary relations indicated by O. For example, a $(+1_l,+1_p,\leq_p)$-structure has some unary relations and a linear order, a preorder successor and its corresponding preorder. An O-structure is a structure from FinOrd(O).

A *word* w over an alphabet $\Sigma = \{\sigma_1,\dots,\sigma_k\}$ is a finite sequence $\tau_1\dots\tau_n$ of letters from Σ. One can think of w as a linear order over $[n]$ where each element i is labeled by letter τ_i from Σ. Thus there is a natural correspondence between words and $\leq_l$-structures (or, alternatively, $+1_l$-structures or $(+1_l,\leq_l)$-structures). Also every $+1_l$-structure naturally corresponds to some word.

Note that words over alphabet $\Sigma = \{\sigma_1,\dots,\sigma_k\}$ correspond to $+1_l$-structures with unary relations $\mathcal{P} = (P_{\sigma_1},\dots,P_{\sigma_k}\}$. On the other hand, $+1_l$-structures with unary relations $\mathcal{P}$ correspond to words over alphabet $2^{\mathcal{P}}$. Here, and in the following, we will ignore this and assume that appropriate alphabets and unary relations are chosen when necessary.

A *preorder word* w is a sequence $\vec{v}_1\dots\vec{v}_l$ of tuples from $\mathbb{N}^{\Sigma}$. A preorder word w can be identified with a preorder $\leq_p$ with Σ-labeled elements where each $\vec{v}_i = (n_{\sigma_1},\dots,n_{\sigma_k})$ is identified with one equivalence class C_i of $\leq_p$. The class C_i contains $\sum_j n_{\sigma_j}$ many elements and n_{σ_j} of those elements are labeled σ_j. Thus a preorder word can be thought of as a word where every position can contain several elements (as opposed to one element in usual words). The identification of tuples with equivalence classes allows for reusing notions for preorders in the context of preorder words, by thinking of $\vec{v}_i$ as an equivalence class. For example, we will say say that $\vec{v}_i$ contains a σ_i-labeled element u, if $n_{\sigma_i} > 0$. Note that there is a natural correspondence between preorder words and ordered $+1_p$-structures (or, alternatively, $\leq_p$-structures or $(+1_p,\leq_p)$-structures).

Ordered Data Words. Fix a finite alphabet $\Sigma = \{\sigma_1,\dots,\sigma_k\}$ and an infinite set $\mathbb{D}$ of *data values* (the *data domain*) which is totally ordered by a linear order $\leq_l^{\mathbb{D}}$. For the purpose of this paper, it is sufficient to think of $\mathbb{D}$ as the set $\mathbb{N}$ of natural numbers and of $\leq_l^{\mathbb{D}}$ as the natural order on $\mathbb{N}$.

An *ordered data word* w is a sequence of pairs from $\Sigma\times\mathbb{D}$. We introduce some important notions for ordered data words. In the following fix an ordered data word $w = (\sigma_1,d_1)\dots(\sigma_n,d_n)$. A preorder $\leq_p$ on $[n]$ is induced by the data values of w by $i \leq_p j$ if $d_i \leq_l^{\mathbb{D}} d_j$. A *class* of w is an equivalence class of $\leq_p$, i.e. a maximal subset $C \subseteq [n]$ of positions of w such that $d_i = d_j$ for all $i,j \in C$. Let, in the following, $C_1 \leq_p \dots \leq_p C_l$ be the classes of w. The *string projection* of w is the word $\sigma_1\dots\sigma_n$ over Σ and is denoted by *sp*(w). The *preorder projection pp*(w) is the preorder word that corresponds to $\leq_p$, that is $pp(w) = \vec{c}_1\dots\vec{c}_l$ where each $\vec{c}_i = (n_{\sigma_1},\dots,n_{\sigma_k})$ with n_{σ_j} is the number of σ_j-labeled elements in C_i. Ordered data words naturally correspond to $(+1_l,+1_p,\leq_p)$-structures (again with many alternative representations). See Figure 1 for an example.

Two-Variable Logic on Ordered Structures. Existential monadic second order logic EMSO extends predicate logic by existential quantification of unary relations. The two-variable fragment of EMSO, denoted by EMSO^2, contains all EMSO-formulas whose first-order part uses at most two distinct variables x and y. Two-variable logic FO^2 is the restriction of first order logic to formulas with at most two distinct variable x and y.

Denote by $\text{EMSO}(O)$ existential monadic second order logic over a vocabulary that contains some unary relation symbols and binary relation symbols from O which have to be interpreted by O-structures. For example, formulas in $\text{EMSO}(+1_l)$ can use some unary relation symbols and the binary relation symbol $+1_l$, and $+1_l$ has to be interpreted as a linear successor. Similar notation will be used for FO^2.

Words, that is $+1_l$-structures, can be seen as interpretations for $\text{EMSO}(+1_l)$-formulas. Similarly preorder words and ordered data words are interpretations for $\text{EMSO}(+1_p, \leq_p)$- and $\text{EMSO}(+1_l, +1_p, \leq_p)$-formulas, respectively.

The language $L(\varphi)$ of $\varphi \in \text{EMSO}^2(+1_l)$ is the set of words, more precisely their corresponding $+1_l$-structures, that satisfy φ. Similarly for other sets of relations. The classical theorem of Büchi, Elgot and Trakhtenbrot states that $\text{EMSO}(+1_l, \leq_l)$ is equivalent to finite state automata. This holds even for $\text{EMSO}^2(+1_l)$. In the next section we introduce an automaton model which is equivalent to $\text{EMSO}^2(+1_l, +1_p, \leq_p)$.

▶ **Example 1.** Let L_1 be the language that contains all data words w over $\Sigma = \{a, b\}$ such that the data value of every a-labeled position in w is smaller than the data values of all b-labeled positions. Let L_2 be the language that contains all data words w such that the a-labeled elements with the largest data value are immediately to the left of a b-labeled element. Then the following $\text{EMSO}^2(+1_l, +1_p, \leq_p)$-formulas φ_1 and φ_2 define L_1 and L_2:

$$\varphi_1 = \forall x \forall y \big((a(x) \wedge b(y)) \rightarrow (x \leq_p y \wedge \neg y \leq_p x)\big)$$
$$\varphi_2 = \forall x \Big(\big(a(x) \wedge \neg\exists y(a(y) \wedge (x \leq_p y \wedge \neg y \leq_p x))\big) \rightarrow \exists y \big(b(y) \wedge +1_l(x, y)\big)\Big)$$

3 An Automaton Model for Ordered Data Words

In this section we introduce ordered data automata, an automaton model for structures with one linear successor relation $+1_l$ (of an underlying linear order $\leq_l$) and one preorder relation $\leq_p$ accompanied by its successor relation $+1_p$. This automaton model is an adaption of data automata as introduced in [3]. In the next section ordered data automata are shown to be equivalent to $\text{EMSO}^2(+1_l, +1_p, \leq_p)$.

Very roughly, ordered data automata process a $(+1_l, \leq_p, +1_p)$-structure by reading it once in linear-order-direction and once in preorder-direction. Therefore an essential part of an ordered data automaton is an automaton capable of reading preorder words. We introduce an automaton model for preorder words first.

Preorder Automata. Roughly speaking, preorder automata are finite state automata that read preorder words $w = \vec{w}_1 \ldots \vec{w}_n$. When reading some $\vec{w}_i$, a transition of such an automaton can be applied if the transition matches the current state and the components of $\vec{w}_i$ satisfy interval constraints specified by the transition. We formalize this.

An *interval* $I = (l, r)$ where $l \in \mathbb{N}$ and $r \in \mathbb{N} \cup \{\infty\}$ contains all $i \in \mathbb{N}$ with $l \leq i < r$. A Σ-*constraint* $\vec{c}$ assigns an interval to every $\sigma \in \Sigma$, i.e. it is a tuple from $(\mathbb{N}, \mathbb{N} \cup \{\infty\})^\Sigma$. A tuple $\vec{w} \in \mathbb{N}^\Sigma$ *satisfies* a Σ-constraint $\vec{c}$, if every component n_σ of $\vec{w}$ is in the interval (l, r) asigned to σ by $\vec{c}$.

A *preorder automaton* $\mathcal{A}$ is a tuple $(Q, \Sigma, \Delta, q_I, F)$, where the states Q, the input alphabet Σ, the initial state $q_I \in Q$ and the final states $F \subseteq Q$ are as in usual finite state automata. The transition relation Δ is a finite subset of $Q \times C \times Q$ where C is a set of Σ-constraints.

The semantics is as follows. When p is a state of $\mathcal{A}$ and $\vec{w}$ is a letter from $\mathbb{N}^\Sigma$, then a transition $(p, \vec{c}, q) \in \Delta$ can be applied if $\vec{w}$ satisfies $\vec{c}$. A run of the automaton $\mathcal{A}$ over a word $\vec{w}_1 \dots \vec{w}_n$ is a sequence of transitions $\delta_1 \dots \delta_n$ with $\delta_i = (p_{i-1}, \vec{c}_i, p_i)$ such that δ_i is applicable to $\vec{w}_i$. The run is accepting if $p_0 = q_I$ and $p_n \in F$. The language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of all preorder words with an accepting run of $\mathcal{A}$.

▶ **Example 2.** Let L be the language of preorder words w over $\Sigma = \{a, b\}$ where every letter $\vec{w}_i$ of w contains an a-labeled element and at most two b-labeled elements. The preorder automaton $\mathcal{A}$ with two states s and e, transitions $\{(s, ((1, \infty), (0, 3)), s), (s, ((0, 1), (0, \infty)), e), (s, ((0, \infty), (3, \infty)), e)\}$, initial state s and single finite state s accepts L.

Preorder automata can be seen as a normal form of finite state automata over the product Boolean algebra of finite and cofinite subsets of $\mathbb{N}$ This observation yields immediately:

▶ **Lemma 3.** *Preorder automata are closed under union, intersection, complementation and letter-to-letter projection.*

The Theorem of Büchi, Elgot and Trakhtenbrot translates to preorder automata. The proof is along similar lines.

▶ **Theorem 4.** *For a language L of preorder words, the following statements are equivalent:*
- *There is a preorder automaton that accepts L.*
- *There is an* $\mathrm{EMSO}^2(+1_p)$*-formula that defines L.*

Ordered Data Automata. The *marked string projection* of an ordered data word is its string projection annotated by information about the relationship of data values of adjacent positions. Formally, let $w = (\sigma_1, d_1) \dots (\sigma_n, d_n)$ be an ordered data word. Then the *marking* $m(i) = (m, m')$ of position i is a tuple from $\Sigma_M = \{-\infty, -1, 0, 1, \infty, -\}^2$ and is defined as follows. If $i = 1$ (or $i = n$) then $m = -$ (or $m' = -$). Otherwise let $C_1 \leq_p \dots \leq_p C_r$ be the classes of w. If C_k, C_l and C_s are the classes of d_{i-1}, d_i and d_{i+1}, respectively, then

$$m(i) = \begin{cases} -\infty & \text{if} \quad l > k+1 \\ -1 & \text{if} \quad l = k+1 \\ 0 & \text{if} \quad l = k \\ 1 & \text{if} \quad l = k-1 \\ \infty & \text{if} \quad l < k-1 \end{cases} \qquad m'(i) = \begin{cases} -\infty & \text{if} \quad l > s+1 \\ -1 & \text{if} \quad l = s+1 \\ 0 & \text{if} \quad l = s \\ 1 & \text{if} \quad l = s-1 \\ \infty & \text{if} \quad l < s-1 \end{cases}$$

The marked string projection of w is the string $(\sigma_1, m(1)) \dots (\sigma_n, m(n))$ over $\Sigma \times \Sigma_M$ and is denoted by $msp(w)$.

An *ordered data automata* (short: ODA) $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ over Σ consists of a non-deterministic letter-to-letter finite state transducer (short: string transducer) $\mathcal{B}$ with input alphabet $\Sigma \times \Sigma_M$ and output alphabet Σ', and a preorder automaton $\mathcal{C}$ with input alphabet Σ'.

An ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ works as follows. First, for a given ordered data word w, the transducer $\mathcal{B}$ reads the marked string projection of w. A run ρ_B of the transducer defines a unique (for each run) new labelling of each position. Let w' be the ordered data word thus obtained from w. Second, the preorder automaton $\mathcal{C}$ runs over the preorder projection of w' yielding a run ρ_C. The run $\rho_{\mathcal{A}} = (\rho_B, \rho_C)$ of $\mathcal{A}$ is accepting, if both ρ_B and ρ_C are accepting. The automaton $\mathcal{A}$ accepts w if there is an accepting run of $\mathcal{A}$ on w. The set of ordered data words accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$.

▶ **Example 5.** The language L_1 from Example 1 can be decided by an ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ with $\Sigma = \Sigma' = \{a, b\}$ as follows. Let w be an ordered data word. The string transducer $\mathcal{B}$ does not relabel any position. Thus the input preorder word of the preorder automaton $\mathcal{C}$ is the preorder projection $\vec{w}_1 \dots \vec{w}_m$ of w. The preorder automaton $\mathcal{C}$ verifies that after the first $\vec{w}_i$ containing an b-labeled element, no a-labeled element occurs in any $\vec{w}_j$ with $j \geq i$.

The language L_2 can be decided by an ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ with $\Sigma = \{a, b\}$ and $\Sigma' = \{a, b\} \times \{0, 1\}$ as follows. Let $w = (\sigma_1, d_1) \dots (\sigma_n, d_n)$ be an ordered data word. The automaton $\mathcal{A}$ processes w as follows. The string transducer $\mathcal{B}$ guesses the a-labeled positions with the largest data value, relabels them with $(a, 1)$ and checks that the following position is b-labeled. All other letters σ are relabeled by $(\sigma, 0)$. Let w' be the ordered data word thus obtained. The input of $\mathcal{C}$ is the preorder projection $\vec{w}'_1 \dots \vec{w}'_m$ of w', and $\mathcal{C}$ verifies that $(a, 1)$-labeled elements occur only in $\vec{w}'_m$.

▶ **Lemma 6.** *Languages accepted by ODA are closed under union, intersection and letter-to-letter projection.*

The following proposition can be proved like Lemma 3 in [24].

▶ **Proposition 1.** *Languages accepted by ODA are not closed under complementation.*

4 Ordered Data Automata and $\mathrm{EMSO}^2(+1_l, +1_p, \leq_p)$ are equivalent

In this section we prove

▶ **Theorem 7.** *For a language L of ordered data words, the following statements are equivalent:*
- *L is accepted by an ordered data automaton.*
- *L is definable in $\mathrm{EMSO}^2(+1_l, +1_p, \leq_p)$.*

This equivalence transfers to the case where the preorder is a linear order (i.e. every equivalence class of the preorder is of size one).

The construction of a formula from an automaton is straightforward. The other direction proceeds by translating a given EMSO^2-formula φ into an equivalent formula in Scott Normal Form, i.e. into a formula of the form $\exists X_1 \dots X_n (\forall x \forall y\ \psi \wedge \bigwedge_i \forall x \exists y\ \chi_i)$ where ψ and χ_i are quantifier-free formulas (see e.g. [13] for the translation). Since ODA are closed under union, intersection and renaming it is sufficient to show that for every formula of the form $\forall x \forall y\ \psi$ and $\forall x \exists y\ \chi$ there is an equivalent ODA.

The proofs of the following lemmas use the abbreviations

$$\begin{array}{lcl}\Delta_= & = & \{x = y, x \neq y\},\\ \Delta_l & = & \{+1_l(x,y), \neg +1_l(x,y), +1_l(y,x), \neg +1_l(y,x)\},\\ \Delta_p & = & \{+1_p(x,y), +1_p(y,x), x \sim_p y, x \ll_p y, y \ll_p x\}.\end{array}$$

▶ **Lemma 8.** *For every formula of the form $\forall x \forall y\ \psi$ with quantifier-free ψ there is an equivalent ODA.*

Proof. We first write ψ in conjunctive normal form and distribute the universal quantifier over the conjunction. Therefore, again due to the closure of ODA under intersection, we can restrict our attention to formulas of the form

$$\varphi = \forall x \forall y (\alpha(x) \vee \beta(y) \vee \delta_=(x,y) \vee \delta_l(x,y) \vee \delta_p(x,y))$$

where α, β are unary formulas and $\delta_=(x,y)$, $\delta_l(x,y)$ and $\delta_p(x,y)$ are as follows. Denote by $\mathsf{Disj}(\Phi)$ the set of disjunctive formulas over a set of formulas Φ. The formulas $\delta_=(x,y)$, $\delta_l(x,y)$ and $\delta_p(x,y)$ are in $\mathsf{Disj}(\Delta_=)$, $\mathsf{Disj}(\Delta_l)$ and $\mathsf{Disj}(\Delta_p)$, respectively. Note that Δ_p contains only positive formulas since negation of any formula in Δ_p can be replaced by a disjunction of formulas from Δ_p.

Without loss of generality we assume that neither $\delta_=(x,y)$, $\delta_l(x,y)$ nor $\delta_p(x,y)$ are the empty disjunction. (Assume that $\delta_=(x,y)=\bot$, then $\delta_=(x,y)\equiv x=y\wedge x\neq y$. Distributing $x=y\wedge x\neq y$ yields two formulas of the required form.)

In the following we do an exhaustive case analysis. If φ is a tautology, then there is an equivalent ODA. Therefore we assume from now on that φ is not a tautology.

When φ is not a tautology then $\delta_=$ is either $x\neq y$ or $x=y$. If $\delta_=$ is $x\neq y$ then we can write φ as $\forall x\forall y\big((\alpha'(x)\wedge\beta'(y)\wedge x=y)\rightarrow\gamma(x,y)\big)$ where α' and β' are the negations of the unary formulas α and β and $\gamma(x,y)=\delta_l(x,y)\vee\delta_p(x,y)$. Substituting $x=y$ in γ yields a formula that is equivalent to True or to False. Thus the property expressed by φ can be checked by the string transducer of an ODA. Hence from now on we assume that $\delta_=$ is the formula $x=y$.

The formula δ_l can either contain a negative formula from Δ_l or it does not contain any negative formula. If δ_l contains a negative formula from Δ_l we rewrite φ as

$$\forall x\forall y\big((\alpha'(x)\wedge\beta'(y)\wedge x\neq y\wedge\delta'_l(x,y))\rightarrow\delta_p(x,y)\big)$$

where δ'_l is the negation of δ_l. Since δ'_l is a conjunction that contains a positive formula from Δ_l it is logically equivalent to a positive formula from Δ_l, that is, it is equivalent either to $+1_l(y,x)$ or to $+1_l(x,y)$. In this case the formula φ expresses a regular property over the marked string projection of the structure. Hence it can be seen immediately that the property expressed by φ can be checked by the string transducer of an ODA. Hence from now on we assume that δ_l contains no negative formula from Δ_l.

Then δ_l is either $+1_l(x,y)\vee+1_l(y,x)$ or $+1_l(y,x)$ or $+1_l(y,x)$. In this case we rewrite φ as $\forall x\forall y\big((\alpha'(x)\wedge\beta'(y)\wedge x\neq y\wedge\delta'_p(x,y))\rightarrow\delta_l(x,y)\big)$ where δ'_p is the negation of $\delta_p(x,y)$. As noted before, the conjunction $\delta'_p(x,y)$ can be expressed as a disjunction of formulas from Δ_p. Hence φ is equivalent to $\forall x\forall y\big((\alpha'(x)\wedge\beta'(y)\wedge x\neq y\wedge\delta''_p(x,y))\rightarrow\delta_l(x,y)\big)$ where $\delta''_p(x,y)$ is a disjunction of formulas in Δ_p. Distributing this disjunction yields a formula of the form $\forall x\forall y\bigwedge\big((\alpha'(x)\wedge\beta'(y)\wedge x\neq y\wedge\delta'''_p(x,y))\rightarrow\delta_l(x,y)\big)$ where $\delta'''_p(x,y)$ is a formula from Δ_p.

By distributing the conjunction over the $\forall$-quantifiers and by using the closure of ODA under intersection, it is sufficient to show that there is an equivalent ODA for formulas of the form $\chi=\forall x\forall y\big((\alpha'(x)\wedge\beta'(y)\wedge x\neq y\wedge\delta_p(x,y))\rightarrow\delta_l(x,y)\big)$ where $\delta_p(x,y)$ is a formula from Δ_p and δ_l is positive.

For the following, we assume that δ_l is the formula $+1_l(x,y)\vee+1_l(y,x)$. The cases $\delta_l=+1_l(x,y)$ and $\delta_l=+1_l(y,x)$ are similar. We do a case analysis for $\delta_p(x,y)$.

Let $\delta_p=+1_p(x,y)$. Assume that C_i and C_{i+1} are two adjacent $\leq_p$-classes. Then the formula χ states that whenever C_i contains an α'-labeled element u and C_{i+1} contains a β'-labeled element v, then u and v are adjacent with respect to $\leq_l$. This implies that the number of α'-labeled elements in C_i and β'-labeled elements in C_{i+1} is at most three. Moreover those elements are adjacent in the linear order.

Thus, an ODA verifying this property can be constructed as follows. The string transducer annotates every α'-labeled element u by the number of β'-labeled elements v with $+1_p(u,v)$ and either $+1_l(u,v)$ or $+1_l(v,u)$. Analogously the string transducer annotates every β'-labeled element by the number of adjacent α'-labeled elements in the preceding $\leq_p$-class.

Then the preorder automaton verifies for each $\leq_p$-class C_i and its successor $\leq_p$-class C_{i+1} that either

- C_i contains no α'-labeled elements or C_{i+1} contains no β'-labeled elements, or
- C_i contains an α'-labeled element and C_{i+1} contains a β'-labeled element and

- C_i and C_{i+1} contain more than three of those elements (then the preorder automaton rejects)
- C_i and C_{i+1} contain less than three of those elements. Then it checks that those three are adjacent by using the annotation given by the transducer (and accepts or rejects accordingly).

The cases $\delta_p = x \sim_p y$ and $\delta_p = x \ll_p y$ are very similar.

◀

▶ **Lemma 9.** *For every formula of the form $\forall x \exists y\ \chi$ with quantifier-free χ there is an equivalent ODA.*

The proof of Lemma 9 will be presented in the full version of the paper. This completes the proof of Theorem 7.

5 Deciding Emptiness for Ordered Data Automata on k-bounded Ordered Data Words

An ordered data word w is k-bounded if each class of w contains at most k elements. In this case the preorder projection of w is a k-bounded preorder word and can be seen as a word over the finite alphabet $\{0, \ldots, k\}^{|\Sigma|}$. Hence an ODA restricted to k-bounded ordered data words can be seen as a composition of a finite state transducer and a finite state automaton. We call such automata *k-bounded ODA*.

Since k-boundedness can be expressed in $\text{EMSO}^2(+1_l, +1_p, \leq_p)$ we can conclude that the result from the previous section carry over to the case of k-bounded ordered data words, i.e. a language $\mathcal{L}$ of k-bounded ordered data words is accepted by a k-bounded ODA if and only if it can be defined by an $\text{EMSO}^2(+1_l, +1_p, \leq_p)$ formula φ.

The rest of this section is devoted to the proof of the following theorem.

▶ **Theorem 10.** *The finite satisfiability problem for $\text{EMSO}^2(+1_l, +1_p, \leq_p)$ on k-bounded data words is decidable.*

▶ **Corollary 11.** *The finite satisfiability problem for $\text{EMSO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$ is decidable.*

This generalizes Theorem 3 from [24], where the finite satisfiability problem of $\text{FO}^2(+1_{l_1}, +1_{l_2})$ was shown to be decidable.

We sketch the proof of Theorem 10; a detailed proof will appear in the full paper. By the above remarks it is sufficient to show that the emptiness problem of k-bounded ODA is decidable. We reduce the emptiness problem for k-bounded ODA to the emptiness problem for multicounter automata. The latter is known to be decidable [25, 22]. The idea is as follows. From a k-bounded ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ we will construct a multicounter automaton $\mathcal{M}$ such that $L(\mathcal{A})$ is non-empty if and only if $L(\mathcal{M})$ is non-empty. Intuitively, $\mathcal{M}$ will be constructed such that if $\mathcal{A}$ accepts a k-bounded ordered data word w then $\mathcal{M}$ accepts a word w' which is the preorder projection of w annotated by lots of extra information[3]. On the other hand if $\mathcal{M}$ accepts an annotated word w' then an ordered data word w and an accepting run of $\mathcal{A}$ on w can be reconstructed from the information encoded in w'. Therefore $\mathcal{M}$ reads a k-bounded preorder word $w' = \vec{w}'_1 \ldots \vec{w}'_m$ and simultaneously verifies

- that the extra information in w' encodes an accepting run of $\mathcal{C}$ on w'.

[3] Recall that the preorder projection of a k-bounded ordered data word is a k-bounded preorder word, i.e. a word over $\{0, \ldots, k\}^{|\Sigma|}$.

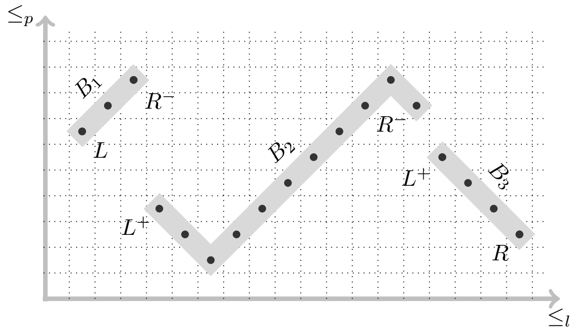


Figure 2 Blocks in the ordered-structure-representation of a 3-bounded ordered data word w. Each • represents one element of w, the $\leq_l$-axis represents positions whereas the $\leq_p$-axis represents data values. Labels are omitted for clarity.

- that the elements occuring in w' can be dynamically (that is while reading $\vec{w}'_1, \vec{w}'_2, \ldots$) arranged to a word x such that x encodes
  - a marked string y whose marking is consistent with w' (and therefore allows for the construction of an ordered data word w from w' and y), and
  - an accepting run of $\mathcal{B}$ on y.

We will need the following notions for ordered data words. A *block* B of an ordered data word w is a maximal subword of w such that all successive positions in B are $\leq_p$-close in w. See Figure 2 for an example of blocks.

Since w is k-bounded, every class of w intersects with at most k many blocks. It is easy to see, that one can color each block B of w with a number $N(B)$ from $\{1, \ldots, 2k\}$ such that $N(B) \neq N(B')$ if B and B' are $\leq_l$-adjacent blocks or $\leq_p$-adjacent blocks. Even more, such a coloring can be uniquely obtained from w (for example by coloring lexicographically).

In the following we describe how to annotate every element of an ordered data word w with extra information. A *block label* (N, X) with *block number* N and *block position* X is a letter from $\Sigma_B = \{1, \ldots, 2k\} \times (\{L, L^+, L^-, C\} \times \{R, R^+, R^-, C\})$. Let $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ be a k-bounded ODA with input alphabet Σ, intermediate alphabet Σ' and let $Q_\mathcal{B}$ and $Q_\mathcal{C}$ be the states of $\mathcal{B}$ and $\mathcal{C}$ respectively. A *run label* $(\sigma', r_\mathcal{B}, r_\mathcal{C}, r_B)$ is a letter from $\Sigma_R = \Sigma' \times Q_\mathcal{B}^2 \times Q_\mathcal{C}^2 \times Q_\mathcal{B}^2$ where $r_\mathcal{B}$, $r_\mathcal{C}$ and b_B are called $\mathcal{B}$*-label*, $\mathcal{C}$*-label* and $\mathcal{B}$*-block label*, respectively.

An *annotated ordered data word* is an ordered data word over $\Sigma \times \Sigma_M \times \Sigma_B \times \Sigma_R$ where Σ_M is the alphabet $\{-\infty, -1, 0, 1, \infty, -\}^2$ of markings. Likewise an *annotated preorder word* is a preorder word over $\Sigma \times \Sigma_M \times \Sigma_B \times \Sigma_R$. The preorder projection of an annotated data word is a preorder word over $\Sigma \times \Sigma_M \times \Sigma_B \times \Sigma_R$.

The *annotation* $\mathsf{ann}(w, \rho)$ of an ordered data word $w = w_1 \ldots w_n$ with respect to a run $\rho = (\rho_\mathcal{B}, \rho_\mathcal{C})$ of an ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ on w is an annotated ordered data word that labels every element w_i with its marking m; a block label τ according to the position of w_i in its block; and a run label π describing the output of $\mathcal{B}$ on run ρ when reading w_i, the states of $\mathcal{B}$ and $\mathcal{C}$ according to run ρ, and the states where $\mathcal{B}$ enters and leaves the block of w_i in run ρ. The preorder projection of the annotation of an ordered data word w is denoted by $\mathsf{annpp}(w, \rho)$.

Intuitively maximal contiguous subwords of $\mathsf{annpp}(w, \rho)$ with the same block number N correspond to a block in w. Therefore such contiguous subwords of annotated preorder words are called *symbolic N-blocks*.

We now state the proof idea of Theorem 10 more precisely. From an ordered data automaton $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ we construct a multicounter automaton $\mathcal{M}$ that reads annotated k-bounded preorder words such that

- If $\mathcal{A}$ accepts a k-bounded ordered data word w via run ρ then $\mathcal{M}$ accepts $\mathsf{annpp}(w, \rho)$.
- If $\mathcal{M}$ accepts an annotated k-bounded preorder word w' then a k-bounded ordered data word w can be constructed from w' which is accepted by $\mathcal{A}$.

Given an annotated k-bounded preorder word $w' = \vec{w}'_1 \dots \vec{w}'_n$, the multicounter automaton $\mathcal{M}$ tries to reconstruct a k-ordered data word w from w' such that w is accepted by $\mathcal{A}$. Every symbolic block B' in w' will represent a block B in w. We will prove that such a reconstruction is possible whenever the following conditions (C0) – (C3) are satisfied:

(C0) a) The block position label and the label from Σ_M are consistent for every element of w'.
b) Every symbolic block B' of w' contains exactly one $\{L, L^-, L^+\}$-labeled element and one $\{R, R^-, R^+\}$-labeled element.
c) All elements of a letter $\vec{w}'_i$ have the same $\mathcal{C}$-label..
d) The $\mathcal{B}$- and $\mathcal{C}$-labels are consistent with the Σ- and Σ'-labels for every element u of w'.

(C1) The $\mathcal{C}$-labels in w' encode an accepting run of $\mathcal{C}$.

(C2) For every symbolic block $B' = \vec{w}'_l \dots \vec{w}'_m$ of w' there is an annotated ordered data word B with data values from the set $\{l, \dots, m\} \subset \mathbb{N}$ such that
a) B is a single block and $pp(B) = B'$. Further, the data value of an element u of B is d when u corresponds to an element contained in $\vec{w}'_d$ in B'.
b) The first position of B carries block position label L, L^+ or L^-. The last position of B carries block position label R, R^+ or R^-. All other positions carry block position label C.
c) All elements of B' carry the same $\mathcal{B}$-block label (p, q).
d) There is a run of $\mathcal{B}$ on B that starts in p, ends in q and is consistent with the $\mathcal{B}$-labels of B.

(C3) Let $B'_1, \dots, B'_m$ be the symbolic blocks of w'. Further let $\vec{w}'_{s_i}$ be the position of B'_i, that contains[4] the $\{L, L^-, L^+\}$-labeled element l_i of B'_i. Analogously let $\vec{w}'_{t_i}$ be the position of B'_i, that contains the $\{R, R^-, R^+\}$-labeled element r_i of B'_i. There is a permutation π of $\{1, \dots, m\}$ such that
a) If (p, q) is the $\mathcal{B}$-block label of $l_{\pi(1)}$, then p is the start state of $\mathcal{B}$. Further the block position label of $l_{\pi(1)}$ is L.
b) If (p, q) is the $\mathcal{B}$-block label of $r_{\pi(m)}$ then q is a final state of $\mathcal{B}$. Further the block position label of $r_{\pi(m)}$ is R.
c) If (p, q) and (p', q') are the $\mathcal{B}$-block labels of $B'_{\pi(i)}$ and $B'_{\pi(i+1)}$, respectively, then $q = p'$.
d) If r_i is labeled with R^+, then l_{i+1} is labeled with L^-. Further $t_i \ll_p s_{i+1}$.
e) Likewise if r_i is labeled with R^-, then l_{i+1} is labeled with L^+. Further $s_{i+1} \ll_p t_i$.

Intuitively, the Conditions (C2) help to reconstruct runs from $\mathcal{C}$. Runs of $\mathcal{B}$ are reconstructed with the help of Conditions (C2) and (C3), where (C2) helps reconstructing runs of $\mathcal{B}$ on blocks whereas (C3) helps reconstructing the order of blocks.

Recall that k-bounded preorder words over Σ can be seen as a word over the finite alphabet $\{0, \dots, k\}^{|\Sigma|}$.

▶ **Lemma 12.** *For every k-bounded ODA $\mathcal{A}$ there is a finite state automaton $\mathcal{M}$ that accepts exactly the annotated k-bounded preorder words that satisfy conditions (C0) and (C1) from above.*

▶ **Lemma 13.** *For every k-bounded ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ there is a finite state automaton $\mathcal{M}$ that accepts exactly the annotated k-bounded preorder words that satisfy condition (C2).*

▶ **Lemma 14.** *For every k-bounded ODA $\mathcal{A}$ there is a multicounter automaton $\mathcal{M}$ that accepts exactly the annotated k-bounded preorder words that satisfy conditions (C3).*

[4] Recall that $\vec{w}'_{s_i}$ can be identified with the equivalence class of the preorder corresponding to B'_i.

Using the previous lemmata we can now complete the proof of Theorem 10.

Proof (of Theorem 10). For a given k-bounded ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ let $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ be the multicounter automata from Lemmata 12, 13 and 14, respectively. Let M be the intersection multicounter automaton for those three automata.

We prove that $L(\mathcal{A})$ is empty if and only if $L(\mathcal{M})$ is empty. The statement of Theorem 10 follows from this. First, let w be a k-bounded ordered data word accepted by $\mathcal{A}$. Then there is an accepting run $\rho = (\rho_{\mathcal{B}}, \rho_{\mathcal{C}})$ of $\mathcal{A}$ on w. The word $w' = \mathsf{annpp}(w, \rho)$ satisfies conditions (C0) – (C3) and is therefore accepted by $\mathcal{M}$ due to Lemmata 12, 13 and 14.

Second, let $w' = \vec{w}'_1 \dots \vec{w}'_m$ be a k-bounded preorder word accepted by $\mathcal{M}$. We construct a k-bounded data word $w \in L(\mathcal{A})$ and an accepting run $\rho = (\rho_{\mathcal{B}}, \rho_{\mathcal{C}})$ of $\mathcal{A}$ on w with $\mathsf{annpp}(w, \rho) = w'$. Therefor let $B'_1, \dots, B'_l$ be the symbolic blocks of w'. Condition (C2) guarantees the existence of annotated data words $B_1, \dots, B_l$ with $pp(B_i) = B'_i$ and data values from $\{l_i, \dots, r_i\}$ when $B'_i = w'_{l_i} \dots w'_{r_i}$. By Condition (C2d) there is a run ρ_i for each B_i starting in p_i and ending in q_i where (p_i, q_i) is the $\mathcal{B}$-label of B'_i.

Now let π the permutation from Condition (C3). We define the ordered data word $w = D_{\pi(1)} \dots D_{\pi(l)}$ where $D_{\pi(i)}$ is obtained from $B_{\pi(i)}$ by removing the annotations. Note that the $D_{\pi(i)}$ are blocks by Conditions (C2a), (C2b), (C3d) and (C3e). The concatenation $\rho_{\mathcal{B}}$ of the runs $\rho_{\pi(1)}, \dots, \rho_{\pi(l)}$ is an accepting run of $\mathcal{B}$ on w by Conditions (C3a), (C3b) and (C3c). An accepting run of $\mathcal{C}$ on the output of $\rho_{\mathcal{B}}$ exists by Condition (C1).

6 Hardness Results for Two-Dimensional Ordered Structures

This section aims at filling the remaining gaps for finite satisfiability of two-variable logic on two-dimensional ordered structures. We refer the reader to Figure 3 for a summary of the results obtained in the literature and here.

We start with a matching lower bound for the finite satisfiability problem of $\mathrm{EMSO}^2(+1_l, +1_p, \leq_p)$ over k-bounded structures. This bound already holds for $\mathrm{FO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$.

▶ **Theorem 15.** *Finite satisfiability of* $\mathrm{FO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$ *is at least as hard as the emptiness problem for multicounter automata.*

▶ **Corollary 16.** *Finite satisfiability of* $\mathrm{FO}^2(+1_l, +1_p, \leq_p)$ *over k-bounded ordered data words is at least as hard as the emptiness problem for multicounter automata.*

It is not surprising that the finite satisfiability problem of FO^2 with two additional preorder successor relations is undecidable, as those allow for encoding a grid. A minor technical difficulty arises when the corresponding equivalence relations are not available. Undecidability even holds for 2-bounded preorder successor relations.

▶ **Theorem 17.** *Finite satisfiability of two-variable logic with two additional 2-bounded preorder successor relations is undecidable.*

We denote the relation ${+1_l}^2$ by $+2_l$. The following slightly improves Theorem 4 in [24].

▶ **Corollary 18.** *Finite satisfiability of* $\mathrm{FO}^2(+1_{l_1}, +2_{l_1}, +1_{l_2}, +2_{l_2})$ *is undecidable.*

The following theorems complement results from [3] and [29]. The proofs use similar methods as used in those works.

▶ **Theorem 19.** *Finite satisfiability of* $\mathrm{FO}^2(+1_l, \leq_l, +1_p)$ *is undecidable.*

▶ **Theorem 20.** *Finite satisfiability of* $\mathrm{FO}^2(+1_{p_1}, \leq_{p_2})$*, i.e. two-variable logic with one additional preorder successor relation and one additional preorder relation, is undecidable.*

7 Discussion

The current status of research on two-variable logic with additional successor and order relations is summarized in Figure 3.

Logic	**Complexity** (lower/upper)	**Comments**
One linear order		
$\mathrm{FO}^2(+1_l)$	NExpTime-complete	[10]
$\mathrm{FO}^2(\leq_l)$	NExpTime-complete	[27, 10]
$\mathrm{FO}^2(+1_l, \leq_l)$	NExpTime-complete	[10]
One total preorder		
$\mathrm{FO}^2(+1_p)$	ExpSpace-complete	ExpCorridorTiling
$\mathrm{FO}^2(\leq_p)$	NExpTime/ExpSpace	
$\mathrm{FO}^2(+1_p, \leq_p)$	ExpSpace-complete	[29]
Two linear orders		
$\mathrm{FO}^2(+1_{l_1}, +1_{l_2})$	NExpTime-complete	[24, 11, 6]
$\mathrm{FO}^2(+1_{l_1}, \leq_{l_2})$	NExpTime/ExpSpace	[29]
$\mathrm{FO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$	Multicounter-Emptiness[5]	★, Corollary 11 and Theorem 15
$\mathrm{FO}^2(+1_{l_1}, \leq_{l_1}, \leq_{l_2})$	NExpTime/ExpSpace	[29]
$\mathrm{FO}^2(+1_{l_1}, \leq_{l_1}, +1_{l_2}, \leq_{l_2})$	Undecidable	[24]
Two total preorders		
$\mathrm{FO}^2(+1_{p_1}, +1_{p_2})$	Undecidable	★, Theorem 17
$\mathrm{FO}^2(+1_{p_1}, \leq_{p_2})$	Undecidable	★, Theorem C.2
$\mathrm{FO}^2(\leq_{p_1}, \leq_{p_2})$	Undecidable	[28]
One linear order and one total preorder		
$\mathrm{FO}^2(+1_l, +1_p)$	? (see discussion)	★ Special case: Theorem 10
$\mathrm{FO}^2(+1_l, \leq_p)$	? (see discussion)	★ Special case: Theorem 10
$\mathrm{FO}^2(+1_l, \leq_l, +1_p)$	Undecidable	★, Theorem 19
$\mathrm{FO}^2(+1_l, \leq_l, \leq_p)$	Undecidable	[3]
$\mathrm{FO}^2(+1_l, +1_p, \leq_p)$	? (see discussion)	★, Special case: Theorem 10
$\mathrm{FO}^2(\leq_l, +1_p, \leq_p)$	ExpSpace-complete	[29]
Many orders		
$\mathrm{FO}^2(\leq_{l_1}, \leq_{l_2}, \leq_{l_3})$	Undecidable	[18]
$\mathrm{FO}^2(+1_{l_1}, +1_{l_2}, +1_{l_3})$	?	
$\mathrm{FO}^2(+1_{l_1}, +1_{l_2}, +1_{l_3}, \ldots)$	?	

Figure 3 Summary of results on finite satisfiability of FO^2 with successor and order relations. Cases that are symmetric and where undecidability is implied are omitted. Results in this paper are marked by ★.

We saw that EMSO^2 with a linear order successor, a k-bounded preorder relation and its induced successor relation is decidable.

After submission of this work, the finite satisfiability problem of $\mathrm{FO}^2(+1_l, +1_p)$ has been shown to be undecidable by Thomas Schwentick and the authors of this work [23], but has not been peer reviewed yet. We strongly conjecture that finite satisfiability for the other remaining open case, namely $\mathrm{FO}^2(+1_l, \leq_p)$, is decidable. We are actually working on the details of the proof and plan to include both results into the full version of this paper.

It remains open whether there is some m such that $\mathrm{FO}^2(+1_{l_1}, \ldots, +1_{l_m})$ is undecidable. A method for proving undecidability of $\mathrm{FO}^2(+1_{l_1}, \ldots, +1_{l_m})$ should not extend to $\mathrm{FO}^2(F_1, \ldots, F_m)$ where $F_1, \ldots, F_m$ are binary predicates that are interpreted as permutations. A successor relation $+1_l$ can be seen as a permutation with only one cycle and one label that marks the first element. Finite satisfiability of $\mathrm{FO}^2(F_1, \ldots, F_m)$ is decidable since one can express that some arbitrary interpreted binary predicate R is a permutation by using

[5] Under elementary reductions.

two-variable logic with counting quantifiers which in turn is decidable by [14]. This is an observation by Juha Kontinen.

▶ **Open Question 1.** *Is there an m such that* $\mathrm{FO}^2(+1_{l_1}, \ldots, +1_{l_m})$ *is undecidable?*

Temporal logics on data words have seen much research recently [8, 9, 17]. However, to the best of our knowledge, most of those logics have been restricted in the sense that comparison of data values was only allowed with respect to equality. In [30] a temporal logic that allows for comparing ordered data values was introduced. The authors intend to use the techniques and results obtained for two-variable logic with additional successors and orders to investigate temporal logics on data values that allow more structure on the data value side.

▶ **Open Question 2.** *Are there expressive but still decidable temporal logics on data words with successor and order relations on the data values?*

We conclude with highlighting a small difference in treating successor relations for data words. In this paper, the preorder successor is *complete* in the sense that every element (except for elements contained in the last preorder equivalence class) has a preorder successor. In many data domains, especially in those that are subject to change, it is sufficient to interpret the preorder successor relation with respect to those data values present in the structure. Such domains are for example the words in the English language, ISBN numbers etc.

However, for data words over the natural numbers it can be useful that some data values are not present in a data word, i.e. that the successor relation can be incomplete. As a complete successor relation can be axiomatized given an incomplete successor relation, this is a more general setting. This setting is used in [29].

References

1. Mikoaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):1–48, 2009.
2. Mikolaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Logic*, 12(4):27:1–27:26, July 2011.
3. Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *LICS*, pages 7–16. IEEE Computer Society, 2006.
4. Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem.* Springer Verlag, 2001.
5. Patricia Bouyer. A logical characterization of data languages. *Inf. Process. Lett.*, 84(2):75–85, 2002.
6. Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and trees. In *LICS*, 2013 (To appear).
7. Alonzo Church. A note on the Entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41, 1936.
8. Stéphane Demri, Deepak D'Souza, and Régis Gascon. A decidable temporal logic of repeating values. In *LFCS*, volume 4514 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 2007.
9. Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.
10. Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279 – 295, 2002.
11. Diego Figueira. Satisfiability for two-variable logic with two successor relations on finite linear orders. *CoRR*, abs/1204.2495, 2012.
12. Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

13 Erich Grädel and Martin Otto. On logics with two variables. *Theor. Comput. Sci.*, 224(1-2):73–113, 1999.

14 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *LICS*, pages 306–317, 1997.

15 Joseph Y Halpern and Yoav Shoham. A propositional modal logic of time intervals. *Journal of the ACM (JACM)*, 38(4):935–962, 1991.

16 Ullrich Hustadt, Renate A Schmidt, and Lilia Georgieva. A survey of decidable first-order fragments and description logics. *Journal of Relational Methods in Computer Science*, 1(251-276):3, 2004.

17 Ahmet Kara, Thomas Schwentick, and Thomas Zeume. Temporal logics on words with multiple data values. In *FSTTCS*, volume 8 of *LIPIcs*, pages 481–492. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

18 Emanuel Kieronski. Decidability issues for two-variable logics with several linear orders. In Marc Bezem, editor, *CSL*, volume 12 of *LIPIcs*, pages 337–351. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

19 Emanuel Kieronski, Jakub Michaliszyn, Ian Pratt-Hartmann, and Lidia Tendera. Two-variable first-order logic with equivalence closure. In *LICS*, pages 431–440, 2012.

20 Emanuel Kieronski and Martin Otto. Small substructures and decidability issues for first-order logic with two variables. In *LICS*, pages 448–457, 2005.

21 Emanuel Kieronski and Lidia Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *LICS*, pages 123–132, 2009.

22 S Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 267–281. ACM, 1982.

23 A. Manuel, T. Schwentick, and T. Zeume. A Short Note on Two-Variable Logic with a Linear Order Successor and a Preorder Successor. *ArXiv e-prints*, June 2013.

24 Amaldev Manuel. Two orders and two variables. In *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 513–524, 2010.

25 Ernst W Mayr. An algorithm for the general petri net reachability problem. *SIAM Journal on computing*, 13(3):441–460, 1984.

26 M. Mortimer. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 21:135–140, 1975.

27 Martin Otto. Two variable first-order logic over ordered domains. *J. Symb. Log.*, 66(2):685–702, 2001.

28 Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. In *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 499–513, 2010.

29 Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. *Logical Methods in Computer Science*, 8(1), 2012.

30 Luc Segoufin and Szymon Torunczyk. Automata based verification over linearly ordered data domains. In *STACS*, volume 9 of *LIPIcs*, pages 81–92. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

31 Wieslaw Szwast and Lidia Tendera. FO^2 with one transitive relation is decidable. pages 317–328, 2013.

32 Boris Trakhtenbrot. The impossibilty of an algorithm for the decision problem for finite models. *Doklady Akademii NaukSSR*, 70(2):569–572, 1950.

33 Alan M Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 42(2):230–265, 1936.

34 Yde Venema. Expressiveness and completeness of an interval tense logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990.

APPENDIX

A Extension of Section 4: Ordered Data Automata and $\mathrm{EMSO}^2(+1_l, +1_p, \leq_p)$ are equivalent

For the convenience of the reader we repeat also some context. In proofs that are only partial in the main paper, the start and end of text not occuring in the main paper is marked at the border by ▼▼▼and ▲▲▲.

The proof of Theorem 7 follows from the following Propositions 2 and 3. The construction of a formula from an automaton is straight forward:

▶ **Proposition 2.** *For every ordered data automaton $\mathcal{A}$ there is a formula $\varphi_{\mathcal{A}} \in \mathrm{EMSO}^2(+1_l, +1_p)$ such that $L(\mathcal{A}) = L(\varphi_{\mathcal{A}})$.*

Proof sketch.

Let $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ be an ODA. Let $\Sigma' = \{\sigma_1, \ldots, \sigma_k\}$ be the output alphabet of $\mathcal{B}$ and input alphabet of $\mathcal{C}$.

As usual the formula $\varphi_{\mathcal{A}}$ encodes successful runs. By (a slight variation) of Büchi, Elgot and Trakhtenbrot, there is a formula $\varphi_{\mathcal{B}}$ in $\mathrm{EMSO}^2(+1_l)$ that encodes successful runs $\rho_{\mathcal{B}}$ of the transducer such that the output is encoded in additional unary second order variables $P = \{P_{\sigma_1}, \ldots, P_{\sigma_k}\}$. Furthermore there is a formula $\varphi_{\mathcal{C}}$ in $\mathrm{EMSO}^2(+1_p)$ which encodes successful runs of $\mathcal{C}$ (where the input for C is encoded by the P-predicates), by Theorem 4.

The desired formula $\varphi_{\mathcal{A}}$ then guesses the predicates P and checks consistency with $\varphi_{\mathcal{B}}$ and $\varphi_{\mathcal{C}}$:

$$\varphi_{\mathcal{A}} = \exists P_1 \ldots P_k(\varphi_{\mathcal{B}} \wedge \varphi_{\mathcal{C}}).$$

◀

▶ **Proposition 3.** *For every formula $\varphi \in \mathrm{EMSO}^2(+1_l, \leq_p, +1_p)$ there is an ordered data automaton $\mathcal{A}_\varphi$ such that $L(\varphi) = L(\mathcal{A}_\varphi)$.*

The proof of Proposition 3 proceeds by translating a given EMSO^2-formula φ into an equivalent formula in Scott Normal Form

$$\exists X_1 \ldots X_n \left(\forall x \forall y\ \psi \wedge \bigwedge_i \forall x \exists y\ \chi_i \right)$$

where ψ and χ_i are quantifier-free formulas (see e.g. [13] for the translation). Since ODA are closed under union, intersection and renaming it is sufficient to show that for every formula of the form $\forall x \forall y\ \psi$ and $\forall x \exists y\ \chi$ there is an equivalent ODA.

The following lemmas use the abbreviations

$$\begin{aligned}
\Delta_= &= \{x = y, x \neq y\},\\
\Delta_l &= \{+1_l(x,y), \neg{+1_l}(x,y), +1_l(y,x), \neg{+1_l}(y,x)\},\\
\Delta_p &= \{+1_p(x,y), +1_p(y,x), x \sim_p y, x \ll_p y, y \ll_p x\}.
\end{aligned}$$

Lemma 8. *For every formula of the form $\forall x \forall y\ \psi$ with quantifier-free ψ there is an equivalent ODA.*

Proof. We first write ψ in conjunctive normal form and distribute the universal quantifier over the conjunction. Therefore, again due to the closure of ODA under intersection, we can restrict our attention to formulas of the form

$$\varphi = \forall x \forall y(\alpha(x) \vee \beta(y) \vee \delta_=(x,y) \vee \delta_l(x,y) \vee \delta_p(x,y))$$

where α, β are unary types and $\delta_=(x,y)$, $\delta_l(x,y)$ and $\delta_p(x,y)$ are as follows. Denote by $\mathsf{Disj}(\Phi)$ the set of disjunctive formulas over a set of formulas Φ. The formulas $\delta_=(x,y)$, $\delta_l(x,y)$ and $\delta_p(x,y)$ are in $\mathsf{Disj}(\Delta_=)$, $\mathsf{Disj}(\Delta_l)$ and $\mathsf{Disj}(\Delta_p)$, respectively. Note that Δ_p contains only positive formulas since negation of any formula in Δ_p can be replaced by a disjunction of formulas from Δ_p.

Without loss of generality we assume that neither $\delta_=(x,y)$, $\delta_l(x,y)$ nor $\delta_p(x,y)$ are the empty disjunction. (Assume that $\delta_=(x,y) = \bot$, then $\delta_=(x,y) \equiv x = y \wedge x \neq y$. Distributing $x = y \wedge x \neq y$ yields two formulas of the required form.)

In the following we do an exhaustive case analysis. If φ is a tautology, then there is an equivalent ODA. Therefore we assume from now on that φ is not a tautology.

When φ is not a tautology then $\delta_=$ is either $x \neq y$ or $x = y$. If $\delta_=$ is $x \neq y$ then we can write φ as

$$\forall x \forall y\big((\alpha'(x) \wedge \beta'(y) \wedge x = y) \rightarrow \gamma(x,y)\big).$$

where α' and β' are the complements of the unary types α and β and $\gamma(x,y) = \delta_l(x,y) \vee \delta_p(x,y)$. Substituting $x = y$ in γ yields a formula that is equivalent to True or to False. Thus the property expressed by φ can be checked by the string transducer of an ODA. Hence from now on we assume that $\delta_=$ is the formula $x = y$.

The formula δ_l can either contain a negative formula from Δ_l or it does not contain any negative formula. If δ_l contains a negative formula from Δ_l we rewrite φ as

$$\forall x \forall y\big((\alpha'(x) \wedge \beta'(y) \wedge x \neq y \wedge \delta_l'(x,y)) \rightarrow \delta_p(x,y)\big)$$

where δ_l' is the negation of δ_l. Since δ_l' is a conjunction that contains a positive formula from Δ_l it is logically equivalent to a positive formula from Δ_l, that is it is equivalent either to $+1_l(y,x)$ or to $+1_l(x,y)$. In this case the formula φ expresses a regular property over the marked string projection of the structure. Hence it can be seen immediately that the property expressed by φ can be checked by the string transducer of an ODA. Hence from now on we assume that δ_l contains no negative formula from Δ_l.

Then δ_l is either $+1_l(x,y) \vee +1_l(y,x)$ or $+1_l(y,x)$ or $+1_l(y,x)$. In this case we rewrite φ as

$$\forall x \forall y\big((\alpha'(x) \wedge \beta'(y) \wedge x \neq y \wedge \delta_p'(x,y)) \rightarrow \delta_l(x,y)\big)$$

where δ_p' is the negation of $\delta_p(x,y)$. As noted before, the conjunction $\delta_p'(x,y)$ can be expressed as a disjunction of formulas from Δ_p. Hence φ is equivalent to

$$\forall x \forall y\big((\alpha'(x) \wedge \beta'(y) \wedge x \neq y \wedge \delta_p''(x,y)) \rightarrow \delta_l(x,y)\big)$$

where $\delta_p''(x,y)$ is a disjunction of formulas in Δ_p. Distributing this disjunction yields a formula of the form

$$\forall x \forall y \bigwedge \big((\alpha'(x) \wedge \beta'(y) \wedge x \neq y \wedge \delta_p'''(x,y)) \rightarrow \delta_l(x,y)\big)$$

where $\delta_p'''(x,y)$ is a formula from Δ_p.

By distributing the conjunction over the $\forall$-quantifiers and by using the closure of ODA under intersection, it is sufficient to show that there is an equivalent ODA for formulas of the form

$$\chi = \forall x \forall y \big((\alpha'(x) \wedge \beta'(y) \wedge x \neq y \wedge \delta_p(x,y)) \rightarrow \delta_l(x,y) \big)$$

where $\delta_p(x,y)$ is a formula from Δ_p and δ_l is positive.

For the following, we assume that δ_l is the formula $+1_l(x,y) \vee +1_l(y)$. The cases
▼▼▼ $\delta_l = +1_l(x,y)$ and $\delta_l = +1_l(y,x)$ are similar. We do a case analysis for $\delta_p(x,y)$.

- If $\delta_p = x \sim_p y$ then the formula χ states that whenever an α'-labeled element u and a β'-labeled element v are in the same $\leq_p$-class, then u and v are adjacent with respect to $\leq_l$. Since any element can have at most two adjacent elements with respect to $\leq_l$, this implies that every $\leq_p$-class containing both an α'-labeled element and a β'-labeled element contains at most three elements labeled either α' or β'. Moreover those elements are adjacent in the linear order.
  An ordered data automaton can verify this property in the following way. The string transducer annotates every α'-labeled element u by the number of β'-labeled elements v that are adjacent to u in the linear order and in the same $\leq_p$-class as u, i.e. by the number of v with $\beta'(v)$, $u \sim_p v$ and either $+1_l(u,v)$ or $+1_l(v,u)$. Remember that the string transducer reads the marked string projection of an ordered data word and therefore knows whether elements adjacent via $\leq_l$ are in the same $\leq_p$-class. For the annotation an appropriate intermediate alphabet Σ' is used. Analogously the string transducer annotates every β'-labeled element by the number of adjacent α'-labeled elements in the same $\leq_p$-class.
  Then the preorder automaton verifies for each $\leq_p$-class C that either
  - C contains no α'-labeled elements or no β'-labeled elements, or
  - C contains both α'-labeled and β'-labeled elements and
    * C contains more than three of those elements (then the preorder automaton rejects)
    * C contains less than three of those elements. Then it checks that those three are adjacent by using the annotation given by the transducer (and accepts or rejects accordingly). ▲▲▲
- The case $\delta_p = +1_p(x,y)$ is very similar. Assume that C_i and C_{i+1} are two adjacent $\leq_p$-classes. Then the formula χ states that whenever C_i contains an α'-labeled element u and C_{i+1} contains a β'-labeled element v, then u and v are adjacent with respect to $\leq_l$. This implies that the number of α'-labeled elements in C_i and β'-labeled elements in C_{i+1} is at most three. Moreover those elements are adjacent in the linear order.
  Thus, the construction of an ODA verifying this property is analogous to the previous case. The string transducer annotates every α'-labeled element u by the number of β'-labeled elements v with $+1_p(u,v)$ and either $+1_l(u,v)$ or $+1_l(v,u)$. Analogously the string transducer annotates every β'-labeled element by the number of adjacent α'-labeled elements in the preceding $\leq_p$-class.
  Then the preorder automaton verifies for each $\leq_p$-class C_i and its successor $\leq_p$-class C_{i+1} that either
  - C_i contains no α'-labeled elements or C_{i+1} contains no β'-labeled elements, or
  - C_i contains an α'-labeled element and C_{i+1} contains a β'-labeled element and
    * C_i and C_{i+1} contain more than three of those elements (then the preorder automaton rejects)
    * C_i and C_{i+1} contain less than three of those elements. Then it checks that those three are adjacent by using the annotation given by the transducer (and accepts or rejects accordingly).

- Finally let $\delta_p = x \ll_p y$. For some ordered data word w consider the minimal $\leq_p$-class C_k containing an α'-labeled element (i.e. there is no class $C_{k'}$ containing an α-labeled element with $k' < k$) and the maximal $\leq_p$-class C_l containing a β'-labeled element. There are two cases.

(1) If $l \leq k + 1$ then χ is satisfied.

(2) Otherwise, there are an α'-labeled element $u \in C_k$ and a β'-labeled element $v \in C_l$ with $u \ll_p v$. The formula χ implies that there can be at most two β'-labeled elements $\leq_p$-larger then u and two α'-labeled elements $\leq_p$-smaller then v. Thus there are at most two β'-labeled elements v, v' in the classes $C_{k+2}, \ldots, C_l$ and at most two α'-labeled elements u, u' in the classes $C_k, \ldots, C_{l-2}$.

An ordered data automaton $(\mathcal{B}, \mathcal{C})$ can verify the property χ as follows. The string transducer $\mathcal{B}$ guesses whether Case (1) or Case (2) is satisfied. If Case (1) is guessed, the preorder automaton verifies that this is correct. If Case (2) is guessed, then the string transducer guesses the elements u, u', v, v' (if they exist) and annotates them using colors σ_u $\sigma_{u'}$, σ_v and $\sigma_{v'}$. Further it checks that they occur successively with respect to $\leq_l$ and annotes all of them with the order in which they occur (for example with colors $\sigma_{(u,v',u',v)}$ if $u \leq_l v' \leq_l u'' \leq_l v$). The preorder automaton then verifies that the elements u, u', v, v' have been guessed correctly, i.e. that

- u is the $\leq_p$-smallest α'-labeled element, v is the $\leq_p$-largest β'-labeled element (using the colors σ_u, $\sigma_{u'}$, σ_v and $\sigma_{v'}$).
- u, u', v and v' satisfy the χ (using the colors σ_u, $\sigma_{u'}$, σ_v and $\sigma_{v'}$ and the color $\sigma_{(u,v',u',v)}$)

◀

Lemma 9. *For every formula of the form $\forall x \exists y\ \chi$ with quantifier-free χ there is an equivalent ODA.*

Proof. We first simplify a given formula $\forall x \exists y\ \chi$ similar to [2, Lemma 13]. Denote by $\mathsf{Conj}(\Phi)$ the set of conjunctive formulas over a set of formulas Φ. As a first step we rewrite χ as an exponential-sized conjunction of disjunctions of the form

$$\forall x \exists y \wedge_i \vee_j (\alpha_i(x) \rightarrow \theta_{ij})$$

where $\{\alpha_i\}_i$ is the set of maximal unary types (a maximal unary type is a conjunction of unary literals that contains, for every unary predicate P, either $P(x)$ or $\neg P(x)$) and each θ_{ij} is of the form

$$\beta(y) \wedge \delta_=(x, y) \wedge \delta_l(x, y) \wedge \delta_p(x, y)$$

for some unary type β and formulas $\delta_=(x, y)$, $\delta_l(x, y)$ and $\delta_p(x, y)$ in $\mathsf{Conj}(\Delta_=)$, $\mathsf{Conj}(\Delta_l)$ and $\mathsf{Conj}(\Delta_p)$, respectively. This rewriting can be achieved using the truth table for χ (That is: Consider χ as a propositional formula where the propositional variables are the atoms of χ. It can be easily seen that this propositional formula is equivalent to a formula $\wedge_i(\alpha_i(x) \rightarrow \vee_j \theta_{ij})$ which describes for each $\alpha_i(x)$ which boolean combinations of other atoms are possible. The disjunction can be pulled out of the formula.)

Since $\{\alpha_i\}_i$ are all maximal unary types χ can be further rewritten as $\wedge_i \forall x \exists y \vee_j (\alpha_i(x) \rightarrow \theta_{ij})$. The disjunction can be eliminated by introducing existentially quantified unary predicates S_{ij} for each θ_{ij} expressing the fact that for an x satisfying the premise α_i and S_{ij}, a witness y satisfying θ_{ij} is chosen. Every conjunct $\forall x \exists y \vee_j (\alpha_i(x) \rightarrow \theta_{ij})$ is rewritten as

$$\exists S_{i1} \dots \exists S_{ij} \Big(\forall x \bigvee_j S_{ij}(x)\Big) \wedge \bigwedge_j \forall x \exists y \big((\alpha_i(x) \wedge S_{ij}(x)) \rightarrow \theta_{ij}(x,y)\big)$$

An ODA equivalent to $\forall x \bigvee_j S_{ij}(x)$ is easy to construct since it is a regular property over the string projection. Since ODA are closed under relabelling and intersection it is sufficient to construct an ODA for each formula of the form

$$\varphi = \forall x \exists y (\alpha(x) \rightarrow \theta_{ij}(x,y))$$

In the following we do an exhaustive case analysis. If $\theta_{ij} \equiv \perp$ or $\delta_=(x,y) \equiv x = y$ then the property expressed by φ can be easily checked by the string transducer of an ODA. Therefore from now on we assume that θ_{ij} is not equivalent to $\perp$ and that $\delta_=(x,y)$ is $x \neq y$.

If $\delta_l(x,y)$ contains a positive literal, the property expressed by φ can be verified by the string transducer of an ODA, since the string transducer reads the marked string projection. Hence from now on we assume that $\delta_l(x,y)$ does not contains a positive literal. Thus let us assume that $\delta_l = \neg{+1_l}(x,y) \wedge \neg{+1_l}(y,x)$ (other cases are similar).

We do a case analysis for $\delta_p(x,y)$.

- When $\delta_p = x \sim_p y$ then the formula is true, if every $\leq_p$-class C containing an α-labeled element u contains a β-labeled element which is not a neighbour (in the linear order) of u. If C contains at least three β-labeled elements then every α-labeled element in C has a witnissing β-element.

  Therefore an ordered data automaton $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ can verify the property expressed by φ as follows. The string transducer $\mathcal{B}$ annotates, using an appropiate intermediate alphabet Σ', every α-labeled element u with the number of β-labeled elements v with $u \sim_p v$ and $+1_l(u,v)$ or $+1_l(v.u)$. For doing this $\mathcal{B}$ uses the marking. The preorder automaton verifies for each $\leq_p$-class C that either C contains at least three β-labeled elements or for every α-labeled element in C there is a β which is not $\leq_l$-adjacent to it. The second case is achieved by looking at the annotation given by the string transducer.
- The case when when δ_l is the formula $+1_p(x,y)$ is very similar to the previous case.
- Finally let δ_p be the formula $x \ll_p y$. We first argue that if φ is satisfied for an ordered data word, then there is a set V of β-labeled elements of size at most three, such that every α-labeled element u is witnessed by some $v \in V$, that is $u \ll_p v$ and the elements u and v are not $\leq_l$-adjacent. Let C be the maximal $\leq_p$-class of an ordered data word w that contains an α-labeled element u (i.e. there is no class C' with $C <_p C'$ that contains an α labeled element). If w satisfies φ then there is a β-labeled element v_1 that witnesses u. Now, there are at most two α-labeled elements not witnessed by v_1 (namely α-labeled elements $\leq_l$-adjacent to v_1). Let v_2 and v_3 be witnesses for those elements (if they exists) and $V = \{v_1, v_2, v_3\}$.

  Now, an ordered data automaton $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ can verify the property expressed by φ as follows. The string transducer $\mathcal{B}$ guesses the elements v_1, v_2, v_3 in V (if they exist) and annotates them with σ_{v_1}, σ_{v_2} and σ_{v_3}. Further it annotates every α-labeled element u witnessed by v_i by σ_{v_i}, and ensures that u and v_i are not $\leq_l$-adjacent. Afterwards the preorder automaton verifies for every α-labeled element u annotated with σ_{v_i} that $u \ll_p v_i$.

◀

B Extension of Section 5: Deciding Emptiness for Ordered Data Automata on k-bounded Ordered Data Words

For the convenience of the reader we repeat the whole section. Many definitions from the main paper are made more precise here. Further all proofs can be found here.

An ordered data word w is k-bounded if each class of w contains at most k elements. In this case the preorder projection of w is a k-bounded preorder word and can be seen as a word over the finite alphabet $\{0, \dots, k\}^{|\Sigma|}$. Hence an ODA restricted to k-bounded ordered data words can be seen as a composition of a finite state transducer and a finite state automaton. We call such automata *k-bounded ODA*.

Since k-boundedness can be expressed in $\mathrm{EMSO}^2(+1_l, +1_p, \leq_p)$ we can conclude that the result from the previous section carry over to the case of k-bounded ordered data words;

▶ **Proposition 4.** *For a language $\mathcal{L}$ of k-bounded ordered data words the following are equivalent:*

- *There is a k-bounded ODA $\mathcal{A}$ with $\mathcal{L} = \mathcal{L}(\mathcal{A})$.*
- *There is a sentence $\varphi \in \mathrm{EMSO}^2(+1_l, +1_p, \leq_p)$ such that $\mathcal{L} = \mathcal{L}_k(\varphi)$, where $\mathcal{L}_k(\varphi)$ is the set of all k-bounded ordered data words satisfying φ.*

In this section we will use this proposition to prove the following theorem by showing that emptiness for k-bounded ODA is decidable.

Theorem 10. *The finite satisfiability problem for $\mathrm{EMSO}^2(+1_l, +1_p, \leq_p)$ on k-bounded data words is decidable.*

Since 1-boundedness can be axiomatized in EMSO^2 the following is an immediate corollary.

Corollary 11. *The finite satisfiability problem for $\mathrm{EMSO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$ is decidable.*

This generalizes Theorem 3 from [24], where the finite satisfiability problem of $\mathrm{FO}^2(+1_{l_1}, +1_{l_2})$ was shown to be decidable.

The rest of this section is devoted to prove that the emptiness problem of k-bounded ODA is decidable. Therefore we reduce the emptiness problem for k-bounded ODA to the emptiness problem for multicounter automata. The latter is known to be decidable [25, 22].

We outline our reduction. From a k-bounded ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ we will construct a multicounter automaton $\mathcal{M}$ such that $L(\mathcal{A})$ is non-empty if and only if $L(\mathcal{M})$ is non-empty. Intuitively, $\mathcal{M}$ will be constructed such that if $\mathcal{A}$ accepts a k-bounded ordered data word w then $\mathcal{M}$ accepts a word w' which is the preorder projection of w annotated by lots of extra information[6]. On the other hand if $\mathcal{M}$ accepts an annotated word w' then an ordered data word w and an accepting run of $\mathcal{A}$ on w can be reconstructed from the information encoded in w'. Therefore $\mathcal{M}$ reads a k-bounded preorder word $w' = \vec{w}'_1 \dots \vec{w}'_m$ and simultaneously verifies

- that the extra information in w' encodes an accepting run of $\mathcal{C}$ on w'.
- that the elements occuring in w' can be dynamically (that is while reading $\vec{w}'_1, \vec{w}'_2, \dots$) arranged to a word x such that x encodes
  - a marked string y whose marking is consistent with w' (and therefore allows for the construction of an ordered data word w from w' and y), and
  - an accepting run of $\mathcal{B}$ on y.

[6] Recall that the preorder projection of a k-bounded ordered data word is a k-bounded preorder word, i.e. a word over $\{0, \dots, k\}^{|\Sigma|}$.

Before going into the details we recall multicounter automata and introduce some further notions needed for the proof. Afterwards we proceed with the proof of Theorem 10.

First we introduce multicounter automata, closely following the exposition in [1]. Essentially, a *multicounter automaton* is a finite state automaton equipped with a finite set of counters which can be incremented and decremented. More formally, a multicounter automaton $\mathcal{M}$ is a tuple $(Q, \Sigma, C, \delta, q_I, F)$, where the states Q, the input alphabet Σ, the initial state $q_I \in Q$ and the final states $F \subseteq Q$ are as in usual finite state automata, and C is a finite set (the *counters*). The transition relation δ is a subset of $Q \times (\Sigma \cup \{\epsilon\}) \times \{\text{inc}(c), \text{dec}(c) | c \in C\} \times Q$.

A *configuration* of a multicounter automaton $\mathcal{M}$ is a pair $(p, \vec{n})$ where p is a state and $\vec{n} \in \mathbb{N}^C$ gives a value for each counter in C. Transitions with $\text{inc}(c)$ can be applied always, whereas transitions with $\text{dec}(c)$ can only be applied when $c > 0$. Applying a transition $(p, \sigma, \text{inc}(c), q)$ to a configuration $(p, \vec{n})$ yields a configuration $(q, \vec{n}')$ where n' is obtained from $\vec{n}$ by incrementing counter c and keeping all other counters unchanged. Similarly for transitions $(p, \epsilon, \text{inc}(c), q)$. Analogously, applying a (applicable) transition $(p, \sigma, \text{dec}(c), q)$ to a configuration $(p, \vec{n})$ yields a configuration $(q, \vec{n}')$ where $\vec{n}'$ is obtained from $\vec{n}$ by decrementing counter c. A *run* over a word w is a sequence of configurations consistent with δ. A run is *accepting*, if it starts at configuration $(q_I, \vec{0})$ and ends in some configuration $(q_F, \vec{0})$ with $q_F \in F$. A multicounter automaton accepts a word w if it has an accepting run on w.

We will need the following notions for ordered data words. Fix a k-bounded ordered data word $w = w_1 \ldots w_n$ with $w_i = (\sigma_i, d_i) \ldots (\sigma_n, d_n)$ and induced preorder $\leq_p$. A subword $B = w_l w_{l+1} \ldots w_m$ of w is called a *partial block* if w_i is $\leq_p$-close to w_{i+1} for all $i \in \{l, \ldots, m-1\}$ (i.e. if there is no data value between d_i and $d_i + 1$). A partial block B is a *block* if no larger block contains B that is if (i) $l = 1$ or w_l and w_{l-1} are far away with respect to $\leq_p$, and (ii) $m = n$ or w_m and w_{m+1} are far away with respect to $\leq_p$. The positions w_l and w_m are called *leftmost* and *rightmost* positions of B. Let $B = w_l \ldots w_m$ and $B = w_{l'} \ldots w_{m'}$ be two distinct blocks with classes $C_1 \leq_p \ldots \leq_p C_r$ and $C'_1 \leq_p \ldots \leq_p C'_{r'}$. The blocks B and B' are *$\leq_l$-adjacent* if $+1_l(m, l')$ or $+1_l(m', l)$, and they are *$\leq_p$-adjacent* if their classes are disjunct and $+1_p(C_r, C'_1)$ or $+1_p(C'_{r'}, C_1)$. See Figure 2 for an example of blocks.

Since w is k-bounded, every class of w intersects with at most k many blocks. It is easy to see, that one can color each block B of w with a number $N(B)$ from $\{1, \ldots, 2k\}$ such that $N(B) \neq N(B')$ if B and B' are $\leq_l$-adjacent blocks or $\leq_p$-adjacent blocks. Even more, such a coloring can be uniquely obtained from w (for example by coloring lexicographically).

In the following we describe how to annotate every element of an ordered data word w with extra information. A *block label* (N, X) with *block number* N and *block position* X is a letter from $\Sigma_B = \{1, \ldots, 2k\} \times (\{L, L^+, L^-, C\} \times \{R, R^+, R^-, C\})$. Let $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ be a k-bounded ODA with input alphabet Σ, intermediate alphabet Σ' and let $Q_\mathcal{B}$ and $Q_\mathcal{C}$ be the states of $\mathcal{B}$ and $\mathcal{C}$ respectively. A *run label* $(\sigma', r_\mathcal{B}, r_\mathcal{C}, r_B)$ is a letter from $\Sigma_R = \Sigma' \times Q_\mathcal{B}^2 \times Q_\mathcal{C}^2 \times Q_\mathcal{B}^2$ where $r_\mathcal{B}$, $r_\mathcal{C}$ and b_B are called *$\mathcal{B}$-label*, *$\mathcal{C}$-label* and *$\mathcal{B}$-block label*, respectively.

An *annotated ordered data word* is an ordered data word over $\Sigma \times \Sigma_M \times \Sigma_B \times \Sigma_R$ where Σ_M is the alphabet $\{-\infty, -1, 0, 1, \infty, -\}^2$ of markings. Likewise an *annotated preorder word* is a preorder word over $\Sigma \times \Sigma_M \times \Sigma_B \times \Sigma_R$. The preorder projection of an annotated data word is a preorder word over $\Sigma \times \Sigma_M \times \Sigma_B \times \Sigma_R$.

The *annotation* $\mathsf{ann}(w, \rho)$ of an ordered data word $w = w_1 \ldots w_n$ with respect to a run $\rho = (\rho_\mathcal{B}, \rho_\mathcal{C})$ of an ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ on w is an annotated ordered data word that labels every element w_i with a marking m, an additional block label τ and run label π as follows

- m is the marking $m(i)$ of w_i.
- $\tau = (N(B), X)$ where B is the block of w_i and $X = (X_L, X_R)$ is
  - $X_L = C$ if w_i is neither the leftmost nor the rightmost element of B

  - $X_L = L$ if $i = 1$
  - $X_L = L^+$ if w_i is the leftmost element of B and $C_i \ll_p C_{i-1}$, where C_i and C_{-1} are the classes of w_i and w_{i-1}
  - $X_L = L^-$ if w_i is the leftmost element of B and $C_{i-1} \ll_p C_i$, where C_i and C_{-1} are the classes of w_i and w_{i-1}
  - Analogously for R, R^+ and R^-
- $\pi = (\sigma', (q_{\mathcal{B},i-1}, q_{\mathcal{B},i}), (q_{\mathcal{C},i-1}, q_{\mathcal{C},i}), (q_1, q_2))$ where
  - σ' is the output symbol of the string transducer $\mathcal{B}$ for w_i.
  - $(q_{\mathcal{B},i-1}, q_{\mathcal{B},i})$ are the states of $\mathcal{B}$ before and after reading w_i in the run $\rho_{\mathcal{B}}$
  - $(q_{\mathcal{C},i-1}, q_{\mathcal{C},i})$ are the states of $\mathcal{C}$ before and after reading the class of w_i in the run $\rho_{\mathcal{C}}$
  - (q_1, q_2) are the states of $\mathcal{B}$ before and after reading the block of w_i

The preorder projection of the annotation of an ordered data word w is denoted by $\mathsf{annpp}(w, \rho)$.

Intuitively maximal contiguous subwords of $\mathsf{annpp}(w, \rho)$ with the same block number correspond to a block in w. Therefore we define the following. A *symbolic N-block* of an annotated preorder word $v = \vec{v}_1 \dots \vec{v}_n$ is an annotated preorder word $\vec{v}'_l \dots \vec{v}'_m$ such that

- For all $i \in \{l, \dots, m\}$ the letter $\vec{v}_i$ contains an element labeled with block number N. The letters $\vec{v}_{l-1}$ and $\vec{v}_{m+1}$ (if they exist) contain no element labeled with block number N.
- For all i, the letter $\vec{v}'_i = (n'_{\sigma_1}, \dots n'_{\sigma_r})$ is the projection of $\vec{v}_i = (n_{\sigma_1}, \dots n_{\sigma_r})$ to elements with block number N, that is
  - $n'_{\sigma_j} = n_{\sigma_j}$ for all σ_j with block number N, and
  - $n'_{\sigma_j} = 0$ for all σ_j with a block number different from N.

We now state the proof idea of Theorem 10 more precisely. From an ordered data automaton $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ we construct a multicounter automaton $\mathcal{M}$ that reads annotated k-bounded preorder words such that

- If $\mathcal{A}$ accepts a k-bounded ordered data word w via run ρ then $\mathcal{M}$ accepts $\mathsf{annpp}(w, \rho)$.
- If $\mathcal{M}$ accepts an annotated k-bounded preorder word w' then a k-bounded ordered data word w can be constructed from w' which is accepted by $\mathcal{A}$.

Given an annotated k-bounded preorder word $w' = \vec{w}'_1 \dots \vec{w}'_n$, the multicounter automaton $\mathcal{M}$ tries to reconstruct a k-ordered data word w from w' such that w is accepted by $\mathcal{A}$. Every symbolic block B' in w' will represent a block B in w. We will prove that such a reconstruction is possible whenever the following conditions (C0) – (C3) are satisfied:

(C0) a) The block position label and the label from Σ_M are consistent for every element of w'.
  b) Every symbolic block B' of w' contains exactly one $\{L, L^-, L^+\}$-labeled element and one $\{R, R^-, R^+\}$-labeled element.
  c) All elements of a letter $\vec{w}'_i$ have the same $\mathcal{C}$-label..
  d) The $\mathcal{B}$- and $\mathcal{C}$-labels are consistent with the Σ- and Σ'-labels for every element u of w'.

(C1) The $\mathcal{C}$-labels in w' encode an accepting run of $\mathcal{C}$.

(C2) For every symbolic block $B' = \vec{w}'_l \dots \vec{w}'_m$ of w' there is an annotated ordered data word B with data values from the set $\{l, \dots, m\} \subset \mathbb{N}$ such that
  a) B is a single block and $pp(B) = B'$. Further, the data value of an element u of B is d when u corresponds to an element contained in $\vec{w}'_d$ in B'.
  b) The first position of B carries block position label L, L^+ or L^-. The last position of B carries block position label R, R^+ or R^-. All other positions carry block position label C.
  c) All elements of B' carry the same $\mathcal{B}$-block label (p, q).
  d) There is a run of $\mathcal{B}$ on B that starts in p, ends in q and is consistent with the $\mathcal{B}$-labels of B.

(C3) Let $B'_1, \ldots, B'_m$ be the symbolic blocks of w'. Further let $\vec{w}'_{s_i}$ be the position of B'_i, that contains[7] the $\{L, L^-, L^+\}$-labeled element l_i of B'_i. Analogously let $\vec{w}'_{t_i}$ be the position of B'_i, that contains the $\{R, R^-, R^+\}$-labeled element r_i of B'_i. There is a permutation π of $\{1, \ldots, m\}$ such that
  a) If (p, q) is the $\mathcal{B}$-block label of $l_{\pi(1)}$, then p is the start state of $\mathcal{B}$. Further the block position label of $l_{\pi(1)}$ is L.
  b) If (p, q) is the $\mathcal{B}$-block label of $r_{\pi(m)}$ then q is a final state of $\mathcal{B}$. Further the block position label of $r_{\pi(m)}$ is R.
  c) If (p, q) and (p', q') are the $\mathcal{B}$-block labels of $B'_{\pi(i)}$ and $B'_{\pi(i+1)}$, respectively, then $q = p'$.
  d) If r_i is labeled with R^+, then l_{i+1} is labeled with L^-. Further $t_i \ll_p s_{i+1}$.
  e) Likewise if r_i is labeled with R^-, then l_{i+1} is labeled with L^+. Further $s_{i+1} \ll_p t_i$.

Intuitively, the Conditions (C2) help to reconstruct runs from $\mathcal{C}$. Runs of $\mathcal{B}$ are reconstructed with the help of Conditions (C2) and (C3), where (C2) helps reconstructing runs of $\mathcal{B}$ on blocks whereas (C3) helps reconstructing the order of blocks.

Recall that k-bounded preorder words over Σ can be seen as a word over the finite alphabet $\{0, \ldots, k\}^{|\Sigma|}$. We say that a finite state automaton (resp. multicounter automaton) reads k-bounded preorder words, if its finite alphabet is $\{0, \ldots, k\}^{|\Sigma|}$.

The following lemmata show how a multicounter automata can verify conditions (C0) – (C3). The conditions (C0) – (C2) can even be verified by a finite state automaton.

Lemma 12. *For every k-bounded ODA $\mathcal{A}$ there is a finite state automaton $\mathcal{M}$ that accepts exactly the annotated k-bounded preorder words that satisfy conditions (C0) and (C1) from above.* The proof is straightforward.

A *partial run* r of a finite state transducer $\mathcal{B}$ with states $\mathcal{B}$ is a tuple from $Q_{\mathcal{B}} \times Q_{\mathcal{B}}$. Two partial runs $r = (p, q)$ and $r' = (p', q')$ *connect* to the partial run (p, q') if $q = p'$.

Lemma 13. *For every k-bounded ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ there is a finite state automaton $\mathcal{M}$ that accepts exactly the annotated k-bounded preorder words that satisfy condition (C2).*

Proof sketch. We describe how to construct a k-bounded preorder automaton $\mathcal{N}$ that verifies condition (C2) for a single symbolic block B' of an annotated k-bounded preorder word. The automaton $\mathcal{M}$ can be easily constructed from $\mathcal{N}$ by using the block number in order to identify symbolic blocks. Condition (C2c) can be checked easily, therefore we restrict our attention to conditions (C2a), (C2b) and (C2d).

The following observation is crucial for the construction of $\mathcal{N}$. Let $B' = \vec{w}'_1 \ldots \vec{w}'_m$ be an annotated k-bounded preorder word which is a single symbolic block, i.e. all elements have the same block number. If B' satisfies (C2), then there is an annotated k-bounded block B with classes classes $C_1 <_p \ldots <_p C_m$ as stated in (C2). In particular C_i corresponds to $\vec{w}'_i$. The restriction of $\mathcal{B}$ to elements from $C_1 <_p \ldots <_p C_l$ yields at most $k+1$ partial blocks $B_1, \ldots, B_{k+1}$, for every $l \in \{1, \ldots, m\}$. (Assume that there are more than $k+1$ blocks, then one C_j with $j > l$ has to contain more than k elements.)

The idea for the construction of $\mathcal{N}$ is as follows. After reading the first $l-1$ letters of B' the automaton has constructed, for every B_j from above, an annotated run $\rho_j = (p_j, q_j)$ such that $\mathcal{B}$ reaches B_j in state p_j and leaves B_j in q_j. Those runs (p_j, q_j) are stored in the state of $\mathcal{N}$ together with some information of the leftmost and rightmost letters of each B_j. When reading the lth letter $\vec{w}'_l$ of B', the automaton extends the runs ρ_j according to the

[7] Recall that $\vec{w}'_{s_i}$ can be identified with the equivalence class of the preorder corresponding to B'_i.

annotated letters occurring in $\vec{w}'_l$ (those runs can be merged while reading B', and new runs can occur, but never more than $k+1$ are present). If $\mathcal{N}$ has constructed, after reading B', a single run ρ which is the $\mathcal{B}$-block label of B', then $\mathcal{N}$ accepts. A block B can then be reconstructed from ρ by looking at the construction of ρ.

We turn to a more precise description of $\mathcal{N}$. The automaton $\mathcal{N}$ stores a multiset of at most $k+1$ *annotated runs.* An annotated run is a tuple (r, m) where r is a partial run of $\mathcal{B}$ and m is a marking from Σ_M. Two annotated runs (r, m) and (r', m') *connect* to an annotated run $(r'', m'') = (r, m) \circ (r', m')$ if

- r and r' connect to r'', and
- $m = (X, Y)$, $m' = (X', Y')$, $m'' = (X, Y')$ and Y and X' are compatible (i.e. $Y = 0$ and $X = 0$, or $Y = +1$ and $X = -1$, or $Y = -1$ and $X = +1$)

To every annotated letter $\sigma \in \Sigma \times \Sigma_M \times \Sigma_B \times \Sigma_R$ an annotated run of (r, m) can be assigned where r and m are the $\mathcal{B}$-label and the marking of σ, respectively.

Now we describe how $\mathcal{N}$ processes the symbolic block $B' = \vec{w}'_1 \dots \vec{w}'_m$. Before processing the letter i of B', the automaton $\mathcal{N}$ stores a multiset S_{i-1} of at most $k+1$ annotated runs in its state. The multiset S_0 is empty.

Each multiset S_i contains annotated runs $(r, (X, Y))$ with $X, Y \in \{+1, -, +\infty, -\infty\}$ only. Intuitively, this means that the annotated run r can be extended when reading the next letter of B' (in the case $X = +1$) or the run starts at the left border of the block and thus cannot be resumed (in the case $X \in \{-, +\infty, -\infty\}$. Analogously for Y.

Let R_i be the multiset of at most k annotated runs corresponding to the annotated letters in $\vec{w}'_i$. When reading $\vec{w}'_i$ the automaton $\mathcal{N}$ performs the following steps.

- Extend all annotated runs ρ from S_{i-1} by distinctive runs from R_i. That is for $\rho = (r, (X, Y))$:
  - If $X = +1$ then guess an annotated run $\tau_l \in R_i$ (that has not been chosen so far) such that τ_l and ρ connect to $\rho' = \tau_l \circ \rho$. If no such τ_l exists, then reject.
  - If $Y = +1$ then guess an annotated run $\tau_r \in R_i$ (that has not been chosen so far) such that ρ' and τ_r connect to $\rho'' = \rho' \circ \tau_r$. If no such ρ'' exists, then reject.

  Denote by T_i the set of all runs ρ'' thus obtained. If some annotated run from R_i has not been chosen, then reject.
- If T_i contains connectable runs, then connect them. That is, as long as T_i contains a run $\rho = (r, (X, Y))$ with either $X = 0$ or $Y = 0$:
  - If $Y = 0$ guess a run $\rho' = (r', (X', Y'))$ with $X = 0$ such that ρ' and ρ connect to $\tau = \rho \circ \rho'$ (if no such run ρ' exists, then reject). Replace ρ and ρ' by τ in T_i.
  - If $X = 0$ guess a run $\rho' = (r', (X', Y'))$ with $Y = 0$ such that ρ and ρ' connect to $\tau = \rho' \circ \rho$ (if no such run ρ' exists, then reject). Replace ρ and ρ' by τ in T_i.
- Let S_i be the set of annotated runs T_i thus obtained. Save S_i in the state (if S_i contains more then $k+1$ annotated runs, then reject).

The automaton $\mathcal{N}$ accepts if S_m contains a single run $\rho = (r, (X, Y))$ where r is the $\mathcal{B}$-label of B'.

We outline the correctness. If Condition (C2) is satisfied for a block B', then there is a block B as stated there. The automaton $\mathcal{M}$, while reading B' does its guessing according to B. On the other hand, if there is an accepting run of $\mathcal{M}$ then S_m contains a single run ρ. A block B witnessing Condition (C2) can be reconstructed by looking at how $\mathcal{M}$ constructed the run ρ.

◀

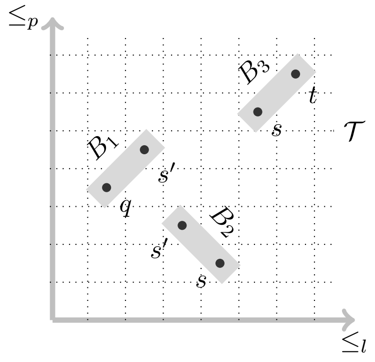


Figure 4 How a multicounter automaton $\mathcal{M}$ reads $\mathsf{annpp}(w, \rho)$ of an annotated data word w. When $\mathcal{M}$ reaches line $\mathcal{T}$, it has stored one partial run (q, s). This run will then be connected to the partial run (s, t) of block B_3.

The construction of a multicounter automaton for (C3) is a little more involved. We introduce some of the ideas for the k-bounded case by warming up with the 1-bounded case. In 1-bounded ordered data words the induced preorder is a linear order, hence equivalence classes can be identified with elements. Further, in 1-bounded annotated data words, the $\{L, L^+, L^-\}$-labeled and $\{R, R^+, R^-\}$-labeled elements are the highest (or lowest) and lowest (or highest) elements of their block. In other words, blocks are merely lines (i.e. without zig-zags, see Figure 4).

▶ **Lemma B.1.** *For every 1-bounded ODA $\mathcal{A}$ there is a multicounter automaton $\mathcal{M}$ that accepts exactly the annotated 1-bounded preorder words that satisfy conditions (C3).*

Proof sketch. Let $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ be an 1-bounded ODA.

The idea for the construction of $\mathcal{M}$ is as follows. Let $w' = B'_1 \dots B'_m$ be an annotated 1-bounded preorder word (i.e. each equivalence class is identified with its single element) with symbolic blocks $B'_1, \dots, B'_m$. Further let ρ_i be the $\mathcal{B}$-block label of B'_i. If there is a permutation as stated in Condition (C3) then while reading the blocks $B'_1, \dots, B'_m$, the automaton $\mathcal{M}$ constructs, for one (guessed) such permutation π, the partial $\mathcal{B}$-run $\rho = \rho_{\pi(1)} \circ \dots \circ \rho_{\pi(m)}$ and verifies that Conditions (C3a)–(C3e) are satisfied.

The automaton $\mathcal{M}$ reads w' block-wise. When $\mathcal{M}$ has read the symbolic blocks $B'_1 \dots B'_i$ then it has stored (in its state and its counters) the set Γ of maximal subruns γ of ρ constructable from the partial runs $\rho_1, \dots, \rho_i$, that is, any subrun $\gamma = \rho_{\pi(j)} \circ \dots \circ \rho_{\pi(j')}$ of ρ with $\{\pi(j), \dots, \pi(j')\} \subseteq \{1, \dots, i\}$ and $\pi(j-1), \pi(j'+1) \notin \{1, \dots, i\}$ is in Γ. Thus every subrun $\gamma = \rho_{\pi(j)} \circ \dots \circ \rho_{\pi(j')}$ stored in Γ corresponds to a reordering $B'_{\pi(j)} \dots B'_{\pi(j')}$ of $B'_j \dots B'_{j'}$. We say that those symbolic blocks *contribute* to γ.

The permutation π will be guessed by $\mathcal{M}$ implicitly. More precisely, when reading a block B'_i containing label L^- (or R^- or both), $\mathcal{M}$ guesses a stored subrun $\gamma \in \Gamma$ that will be attached to ρ_i from the left (since L^- intuitively says, that B'_i has to be attached to another block B'_j with $j < i$). This guessing implicitly determines how the permutation π orders the symbolic block B'_i and the symbolic blocks that contributed to γ.

A partial run ρ_i may be attached to a subrun $\gamma \in \Gamma$ only when their states are consistent. Furthermore, if the last element of B'_{i-1} is R^+ and the first element of B'_i is L^- then the subrun γ of B'_{i-1} may not be attached to the run ρ_i (since then Conditions (C3d) and (C3e) would be injured). Similarly if the last element of B'_{i-1} is R^+ and the first element of B'_i is L^-. Thus, special care has to be taken for the maximal subrun that ρ_{i-1} has contributed to.

If $\mathcal{M}$ has constructed, after reading B', a single run $\rho = (s, q_f)$ where s is the start state and q_f is some final state of $\mathcal{B}$, then $\mathcal{M}$ accepts. The permutation π can then be reconstructed from ρ by looking at the construction of ρ.

We will now turn towards a more detailed description of $\mathcal{M}$. A *block run* is a partial run $\rho = (p, q)$ whose endpoints p and q are marked by some label from $\mathcal{L} \cup \mathcal{R}$, where $\mathcal{L} = \{L, L^-, L^+\}$ and $\mathcal{R} = \{R, R^-, R^+\}$, such that not both are labeled from either $\mathcal{R}$ or $\mathcal{L}$. Two block runs $\rho = (p, q)$ and $\rho' = (p', q')$ can be connected to a block run (p, q') if $q = p'$ and q and p' are marked by either R^+ and L^- or R^- and L^+. A *cached run* is a block run $\rho = (p, q)$ where one of the endpoints p and q is marked with the additional label *locked*. Every symbolic block B' has a corresponding block run (p, q) where B' has block label (p, q) and p is labeled with L^+ (resp. L, L^-) if B has an L^+-labeled element (resp. L, L^--labeled element), analogous for q.

The automaton $\mathcal{M}$ stores block runs by using one counter for every block run from $Q \times Q \times \{\mathcal{L} \cup \mathcal{R}\}^2$, those runs will be referred to as *counted runs*. Furthermore $\mathcal{M}$ can store at most one *cached run* in its state (the maximal subrun to whom the last read block contributed to). We say that a block run ρ *is removed* from the storage of $\mathcal{M}$, if ρ is removed from the cache when ρ is a cached run, or the counter c_ρ is decreased when ρ is a counted run. All runs stored by $\mathcal{M}$ are called *stored runs*.

We describe how $\mathcal{M}$ processes a 1-bounded annotated preorder word $w' = B'_1 \ldots B'_m$ with symbolic blocks $B'_1, \ldots, B'_m$. At the beginning, all counters are zero and no cached run is stored in the state. Now, assume that $\mathcal{M}$ has read $B'_1 \ldots B'_{i-1}$. Let $\rho = (p, q)$ be the block run corresponding to B'_i. Assume, without loss of generality, that the first element[8] and last element of B'_i are labeled with $X_{first} \in \{L, L^+, L^-\}$ and $X_{last} \in \{R, R^+, R^-\}$, respectively (the construction for $X_{first} \in \{R, R^+, R^-\}$ and $X_{last} \in \{L, L^+, L^-\}$ is analogous).

Now, $\mathcal{M}$ performs the following steps:

a) If $X_{first} = L^-$, then $\mathcal{M}$ guesses a stored run γ (from cache or counted) that connects with ρ to $\rho' = \gamma \circ \rho$ (if no such run γ exists, then $\mathcal{M}$ rejects). Then γ is removed from the storage of $\mathcal{M}$. If $X_{first} \neq L^-$ then $\rho' = \rho$.

b) If $X_{last} = R^-$, then $\mathcal{M}$ guesses a stored run γ of $\mathcal{M}$ that connects with ρ' to $\rho'' = \rho' \circ \gamma$ (if no such run γ exists, then $\mathcal{M}$ rejects). Then γ is removed from the storage of $\mathcal{M}$. If $X_{last} \neq R^-$ then $\rho'' = \rho'$.

c) If the cache of $\mathcal{M}$ contains a cached run ρ, remove ρ from the cache and increment the counter c_ρ.

d) If $X_{last} = R^+$ then save $\rho'' = (p, q)$ into the cache with locked q (i.e. the partial run of the next block may not be connected to q).

e) If $X_{last} \neq R^+$ then increment counter $c_{\rho''}$.

The automaton $\mathcal{M}$ accepts, if after reading w' there is a block run $\rho = (s, q_f)$ such that c_ρ has value one, and all other counters have value zero, s is the start state of $\mathcal{B}$ and q_F is some final state of $\mathcal{B}$.

We sketch the correctness of this construction. If $w' = B'_1 \ldots B'_m$ with symbolic blocks $B'_1, \ldots, B'_m$ satisfies Condition (C3), then there is a permutation π as stated. A run of $\mathcal{M}$ can be constructed from w' as follows. The choices in Steps a) and b) are made such that they are consistent with π. I.e. when $\mathcal{M}$ reads $B'_{\pi(i)}$ and the block $B'_{\pi(i-1)}$ has already been read, contributing to the block run γ stored by $\mathcal{M}$, then the partial run of $B'_{\pi(i)}$ is attached (from the right) to the stored run γ.

If, one the other hand, w' is accepted by $\mathcal{M}$ then a permutation π according to (C3) can be constructed from ρ by looking at the choices taken in Steps a) and b).

◀

[8] Recall that in the 1-bounded case equivalence classes are identified with one element.

Lemma 14. *For every k-bounded ODA $\mathcal{A}$ there is a multicounter automaton $\mathcal{M}$ that accepts exactly the annotated k-bounded preorder words that satisfy conditions (C3).*

Proof sketch. We extend the construction of Lemma B.1 and reuse its notation. The situation for k-bounded words w' is more complicated:

- While reading w' several symbolic blocks might have to be processed at once (as opposed to exactly one symbolic block at each moment in the 1-bounded case)
- For a given symbolic block $B' = \vec{w}'_k \dots \vec{w}'_l$ of w', the markers from $\mathcal{L}$ and $\mathcal{R}$ can appear in any $\vec{w}'_i$ and $\vec{w}'_j$ with $i, j \in \{k, \dots, l\}$ (as opposed to only in $\vec{w}'_k$ and $\vec{w}'_l$ in the 1-bounded case).

Those complications are addressed by extending the construction from Lemma B.1 by a cache that stores multiple cached runs. Furthermore, cached runs will store some more information.

Let $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ be a k-bounded ODA. The idea for the construction of $\mathcal{M}$ is similar to the idea in Lemma B.1. Let $w' = w'_1 \dots w'_n$ be an annotated k-bounded preorder word with symbolic blocks $B'_1, \dots, B'_m$ (possibly overlapping, that is, $B'_1 = \vec{w}'_1\vec{w}'_2$ and $B'_2 = \vec{w}'_2\vec{w}'_3$ is possible, see Figure 2) with $\mathcal{B}$-block labels ρ_i. If there is a permutation as stated in Condition (C3) then while reading the blocks w', the automaton $\mathcal{M}$ constructs, for one (guessed) such permutation π, the partial $\mathcal{B}$-run $\rho = \rho_{\pi(1)} \circ \dots \circ \rho_{\pi(m)}$ and verifies that Conditions (C3a)–(C3e) are satisfied.

As in the previous lemma $\mathcal{M}$ will construct a set Γ of maximal subruns γ of ρ constructable from the runs $\rho_1, \dots, \rho_i$ of symbolic blocks $B'_1, \dots B'_i$ seen so far. Yet, since the blocks might overlap, $\mathcal{M}$ cannot read w' strictly block-wise.

For a symbolic block $B' = \vec{w}'_l \dots \vec{w}'_{l'}$, call the smallest $s \in \{l, \dots, l'\}$ such that $\vec{w}'_s$ contains an $\mathcal{L} \cup \mathcal{R}$-labeled element u, the *start position* of B'. The element u is called *start element.* Analogously the largest $t \in \{l, \dots, l'\}$ such that $\vec{w}'_t$ contains an $\mathcal{L} \cup \mathcal{R}$-labeled element u is called *end position* of B', and u is called *end element.* When $\mathcal{M}$ has read a prefix $\vec{w}'_1 \dots \vec{w}'_l$ of w', then a symbolic block with start position s and end position t is called *completed*, if it has been completely read, i.e. if $t < l$. It is called *active* if it has been partially (but not completely) read, i.e. if $s \le l < t$.

Now, when $\mathcal{M}$ has read $\vec{w}'_1 \dots \vec{w}'_j$ then it has constructed the maximal subruns in Γ from the partial runs of active and completed blocks. However, maximal subruns constructed from partial runs of active blocks may not be extended arbitrarily. When $\gamma = \gamma' \circ \rho$ is a maximal subrun build from some other run γ' and a partial run ρ of an active block B', then no partial run of another block may be attached to γ from the right as long as B' is active. Therefore, maximal subruns of still active blocks will be stored in the state of $\mathcal{M}$ (as cached runs).

We will now turn towards a more detailed description of $\mathcal{M}$. Therefore we reuse the notion of *block run* and extend the notion of *cached run* from Lemma B.1. From now on a *cached run* is a block run $\rho = (p, q)$ which is marked by a block number $N \in \{1, \dots, 2k\}$ and whose endpoints p and q are marked by some *status* s, where s is either *closed*, *locked*, *new-locked*, *waiting* or *open.* A cached run (p, q) and a block run (p', q') can be connected to a block run $(p, q) \circ (p', q')$ if (p, q) and (p', q') connect as block runs and q has status open (analogously, (p, q) and (p', q') can be connected to a block run $(p', q') \circ (p, q)$ if p has status open).

The automaton $\mathcal{M}$ stores block runs, as in the previous construction, by using one counter for every block run from $Q \times Q \times \{\mathcal{L} \cup \mathcal{R}\}^2$ (counted runs). Furthermore $\mathcal{M}$ can store at most $4k$ cached runs in its state. Intuitively, a maximal subrun $\gamma \in \Gamma$ is stored as cached run, if in its construction at least one partial run either of a still active block or of a recently completed block was used.

We describe how $\mathcal{M}$ processes a k-bounded annotated preorder word $w' = \vec{w}'_1 \dots \vec{w}'_n$ with symbolic blocks $B'_1, \dots, B'_m$. At the beginning, all counters are zero and no cached run is stored in the state. Now, assume that $\mathcal{M}$ has read $w' = \vec{w}'_1 \dots \vec{w}'_i$.

There can be at most k many $\mathcal{L} \cup \mathcal{R}$-labeled elements in $\vec{w}'_i$. Each such element u corresponds to the start or end position of a symbolic block B'. We will describe how such an element u is processed by $\mathcal{M}$. Assume, without loss of generality, that u is labeled with $X \in \mathcal{L}$ (the construction for $\mathcal{R}$-labeled u is analogous). Further assume, that $\vec{w}'_i$ does not contain another $\mathcal{L} \cup \mathcal{R}$-labeled element v such that u and v have the same block number, i.e. the symbolic block of u does not have the same start and end position (the case where both start and end positions of a block occur in $\vec{w}_i$ is similar).

When the block number of the symbolic block B' of u occurs as block number of some cached run $\gamma \in \Gamma$ then the start position of $\mathcal{B}'$ has been read already, thus i is the end position of B' and u is the end element. Otherwise i is the start position and u the start element. We do a case distinction for those two cases.

When u is the start element of B' then denote the block run of B' by $\rho = (p, q)$. The automaton $\mathcal{M}$ performs the following steps for u:

1) If the $\mathcal{L}$-label X of u is
   a) L^-, then $\mathcal{M}$ guesses a stored run $\gamma = (p', q')$ (if no such run γ exists, then $\mathcal{M}$ rejects) that connects with ρ to a cached run $\rho' = \gamma \circ \rho = (p', q)$ where ρ' is marked as follows
      * If γ is a cached run, then p' is marked as in γ and q is marked waiting.
      * If γ is a counted run, then p' is marked open and q is marked waiting.

      Then γ is removed from the storage of $\mathcal{M}$.
   b) L^+, then $\rho' = (p, q)$ where p is marked as new-locked and q is marked as waiting.
   c) L, then $\rho' = (p, q)$ where p is marked as closed and q is marked as waiting.
2) Store ρ' in the cache of $\mathcal{M}$.

When u is the end element of B' then let $\rho = (p, q)$ be the cached run with the same block number as B'. The automaton $\mathcal{M}$ performs the following steps for u:

1) If the $\mathcal{L}$-label X of u is
   a) L^-, then $\mathcal{M}$ guesses a stored run $\gamma = (p', q')$ (if no such run γ exists, then $\mathcal{M}$ rejects) that connects with ρ to $\rho' = \gamma \circ \rho = (p', q)$. Then
      * If γ is a cached run, then p' is marked as in γ and q is marked open. Further waiting
      * If γ is a counted run, then p' is marked open and q is marked waiting

      Then γ is removed from the storage of $\mathcal{M}$.
   b) L^+, then $\rho' = \rho$ where p is marked as new-locked and q is marked as waiting.
   c) L, then $\rho' = \rho$ where p is marked as closed and q is marked as waiting.
2) Store ρ' in the cache of $\mathcal{M}$.

All $\mathcal{L} \cup \mathcal{R}$-labeled elements u of $\vec{w}'_i$ are processed by M in this way. After this, the automaton $\mathcal{M}$ cleans the cache as follows:

- All cached runs $\rho = (p, q)$ where p and q are neither locked nor waiting are removed from the cache and the counter c_ρ is incremented
- All cached runs $\rho = (p, q)$ where p is marked new-locked are replaced by $\rho' = (p, q)$ where p is marked locked. Analogously when q is marked new-locked.

This ensures that after reading $\vec{w}'_i$ the cache contains at most $2k$ cached runs.

The automaton $\mathcal{M}$ accepts, if after reading w' there is a block run $\rho = (s, q_f)$ such that c_ρ has value one, and all other counters have value zero, s is the start state of $\mathcal{B}$ and q_F is some final state of $\mathcal{B}$.

◀

Using the previous lemmata we can now complete the proof of Theorem 10.

Proof (of Theorem 10). For a given k-bounded ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ let $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ be the multicounter automata from Lemmata 12, 13 and 14, respectively. Let M be the intersection multicounter automaton for those three automata. We prove that $L(\mathcal{A})$ is empty if and only if $L(\mathcal{M})$ is empty. The statement of Theorem 10 follows from this.

First, let w be a k-bounded ordered data word accepted by $\mathcal{A}$. Then there is an accepting run $\rho = (\rho_{\mathcal{B}}, \rho_{\mathcal{C}})$ of $\mathcal{A}$ on w. The word $w' = \mathsf{annpp}(w, \rho)$ satisfies conditions (C0) – (C3) and is therefore accepted by $\mathcal{M}$ due to Lemmata 12, 13 and 14.

Second, let $w' = \vec{w}'_1 \ldots \vec{w}'_m$ be a k-bounded preorder word accepted by $\mathcal{M}$. We construct a k-bounded data word $w \in L(\mathcal{A})$ and an accepting run $\rho = (\rho_{\mathcal{B}}, \rho_{\mathcal{C}})$ of $\mathcal{A}$ on w with $\mathsf{annpp}(w, \rho) = w'$. Therefor let $B'_1, \ldots, B'_l$ be the symbolic blocks of w'. Condition (C2) guarantees the existence of annotated data words $B_1, \ldots, B_l$ with $pp(B_i) = B'_i$ and data values from $\{l_i, \ldots, r_i\}$ when $B'_i = w'_{l_i} \ldots w'_{r_i}$. By Condition (C2d) there is a run ρ_i for each B_i starting in p_i and ending in q_i where (p_i, q_i) is the $\mathcal{B}$-label of B'_i.

Now let π the permutation from Condition (C3). We define the ordered data word $w = D_{\pi(1)} \ldots D_{\pi(l)}$ where $D_{\pi(i)}$ is obtained from $B_{\pi(i)}$ by removing the annotations. Note that the $D_{\pi(i)}$ are blocks by Conditions (C2a), (C2b), (C3d) and (C3e). The concatenation $\rho_{\mathcal{B}}$ of the runs $\rho_{\pi(1)}, \ldots, \rho_{\pi(l)}$ is an accepting run of $\mathcal{B}$ on w by Conditions (C3a), (C3b) and (C3c). An accepting run of $\mathcal{C}$ on the output of $\rho_{\mathcal{B}}$ exists by Condition (C1).

C Extension of Section 6: Hardness Results for Two-Dimensional Ordered Structures

This section aims at filling the remaining gaps for finite satisfiability of two-variable logic on two-dimensional ordered structures. We refer the reader to Figure 3 for a summary of the results obtained in the literature and here.

We start with a matching lower bound for the finite satisfiability problem of $\text{EMSO}^2(+1_l, +1_p, \leq_p)$ over k-bounded structures. This bound already holds for $\text{FO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$.

Theorem 15. *Finite satisfiability of* $\text{FO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$ *is at least as hard as the emptiness problem for multicounter automata.*

Proof. Satisfiability for $\text{FO}^2(\leq_l, +1_l, \sim)$ is at least as hard as reachability in vector addition systems, see [1]. The proof from [1] holds even if $\sim$ is 2-bounded. Since 2-bounded equivalence relations can be simulated by $+1_{l_1}$, the result follows.

For sake of completeness, we sketch the argument from [1]. We reduce the non-emptiness of multicounter automata to satisfiability of $\text{FO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$.

Let $\mathcal{M}$ be a multicounter automaton with counters $C = \{1, \ldots m\}$ and state set Q. The idea is to encode runs of $\mathcal{M}$ as $(+1_{l_1}, +1_{l_2}, \leq_{l_2})$-structures. Therefore, encode transitions of $\mathcal{M}$ as letters from

$$Q \times (\Sigma \cup \{\epsilon\}) \times (\{D_j \mid j \in C\} \cup \{I_j \mid j \in C\}) \times Q$$

where D_j and I_j intuitively stand for decrementing and incrementing the counter j, respectively.

We write a sentence φ in $\text{FO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$ which ensures the following:

- The string projection of the order $\leq_{l_2}$ is of the form $\delta_1 \ldots \delta_n$ with $\delta_i = (p_i, \sigma_i, op_i, q_i)$ such that
  - p_1 is the initial state of $\mathcal{M}$,
  - q_n is a final state of $\mathcal{M}$, and

  - $q_i = p_i$ for $i \in \{1, \ldots, n-1\}$
- The string projection of of the order $\leq_{l_1}$ induced by $+1_{l_1}$ is of the form $\delta'_1 \ldots \delta'_n$ with $\delta'_i = (p'_i, \sigma'_i, op'_i, q'_i)$ such that $op'_1 \ldots op'_n$ is in the language $(I_1 D_1 + \ldots + I_k D_k)^*$.
- It is the case that $\bigwedge_{j \in C} \forall x \forall y \Big((I_j(x) \wedge D_j(y) \wedge +1_{l_1}(x,y)) \rightarrow x \leq_{l_2} y \Big)$.

Intuitively, the second condition states that, in an accepting run, every decrement operation of a counter has exactly one corresponding increment operation and vice versa. The last condition ensures for every decrement operation, the corresponding increment operation occured earlier in the run (and thus a counter cannot obtain a value below zero). ◀

Since the logic $\mathrm{FO}^2(+1_l, +1_p, \leq_p)$ allows for axiomatizing 1-boundedness, by $\forall x \forall y (x \neq y \rightarrow \neg x \sim_p y)$, we have the following corollary.

Corollary 16. *Finite satisfiability of* $\mathrm{FO}^2(+1_l, +1_p, \leq_p)$ *over k-bounded ordered data words is at least as hard as the emptiness problem for multicounter automata.*

It is not surprising that FO^2 with two additional preorder successor relations is undecidable, as those allow for encoding a grid. A minor technical difficulty arises when the corresponding equivalence relations are not available.

▶ **Theorem C.1.** *Finite satisfiability of two-variable logic with two additional preorder successor relations, that is* $\mathrm{FO}^2(+1_{p_1}, +1_{p_2})$*, is undecidable.*

Proof. We reduce from the tiling problem. A *tile* is a square with colored edges. A *valid tiling* of an $m \times n$ grid with tiles from a tile-set T is a mapping $\mathcal{T} : [m] \times [n] \rightarrow T$ such that adjacent edges have the same color, i.e., for example, the northern edge of $\mathcal{T}(i,j)$ and the southern edge of $\mathcal{T}(i, j+1)$ are colored identically. The following tiling problem is undecidable:

Problem:	Tiling
Input:	Tiles $T_1, \ldots, T_k$ over a set of colors $\{c_1, \ldots, c_l\}$.
Question:	Is there a valid tiling of an $m \times n$ grid for some $m, n \in \mathbb{N}$ such that the topside of the top row is colored with c_1 and the bottom side of the bottom row is colored with c_1?

Let I be an instance of Tiling with tiles T. A valid tiling $\mathcal{T} = (\mathcal{T}_{ij})_{i,j \in [m] \times [n]}$ with tiles from T will be encoded by a $(+1_{p_1}, +1_{p_2})$-structure $\mathcal{M}(\mathcal{T})$ with additional unary relation symbols from T and $G = \{W, SW, S, SE, E, NE, N, NW, C\}$ as shown in Figure 5.

We now describe a $\mathrm{FO}^2(+1_{p_1}, +1_{p_2})$-sentence $\varphi = \varphi_{grid} \wedge \varphi_{tiling}$ whose models encode valid tilings. We say that x is northwest of y if $+1_{p_1}(x,y)$ and $+1_{p_2}(y,x)$. Analogously for southwest, southeast and northeast.

For ensuring a grid as seen in Figure 5 it is only necessary that a unique border and all points within this border exist. Thus, the sentence φ_{grid} is the conjunction of a sentence φ_{border} that ensures the correctness of the border and a sentence φ_{center} that ensures the existence of all elements within the border.

Now, φ_{border} is the conjunction of several conditions. Firstly, every label from $\{W, S, E, N\}$ occurs exactly once. Let the elements labeled with W, S, E, N be w, s, e and n. Then for the northwest border, the following conditions need to be satisfied:

- There is a NW-labeled element northeast of w and a NW-labeled element southwest of n.
- For every element labeled with NW there is an element in northeast direction which is labeled with NW or N. Similarly there is an element in southwest direction which is labeled with NW or W.

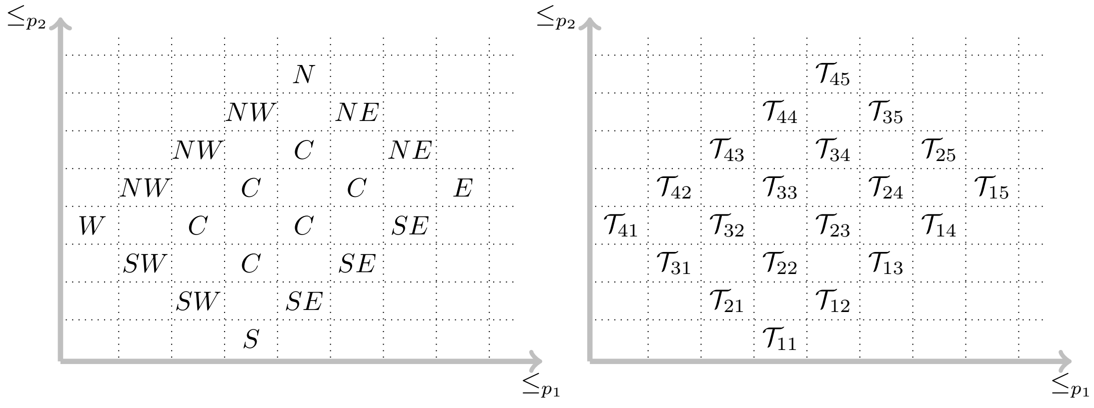


Figure 5 How a tiling $\mathcal{T}_{11}\mathcal{T}_{12}\dots\mathcal{T}_{15}|\dots|\mathcal{T}_{41}\mathcal{T}_{42}\dots\mathcal{T}_{45}$ is encoded as a $(+1_{p_1};+1_{p_2})$-structure. The labels encoding the grid and the tiling are shown on the left and right side, respectively.

Analogously for the southwest, southeast and northeast border.

The formula φ_{center} ensures the following conditions:

- All elements in southeast direction of an NW-labeled element are labeled with C.
- All elements in northwest, southwest, southeast and northeast direction of a C-labeled element are labeled with C, NW, SW, SE or NE, respectively.

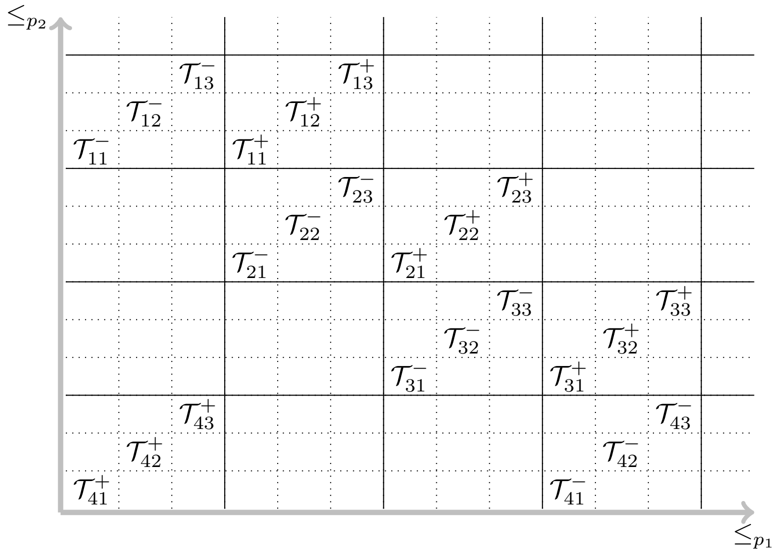
Assuming that the equivalence classes of $+1_{p_1}$ and $+1_{p_2}$ constitute a grid, the formula φ_{tiling} ensures that the grid is tiled validly:

- Every element x carries exactly one label from T.
- Elements labeled with NW carry a tile with c_1-colored top side. Elements labeled with SE carry a tile with c_1-colored bottom side.
- If x is southwest of y, x carries tile s and y carries tile t, then the right side of s and the left side of t are colored equally.

Obviously, if $\mathcal{T}$ is a valid tiling, then $\mathcal{M}(\mathcal{T})$ fulfills φ (see Figure 5). We finish the proof by constructing a valid tiling $\mathcal{T}$ from a model $\mathcal{M}$ of φ. Let $\mathcal{M}$ be a model of φ. Then $\mathcal{M}$ is also a model of φ_{border}, and therefore has unique elements w, s, e, n labeled with W, S, E, N. Further φ_{border} ensures that between w and s, w and n, s and e as well as n and e there are blocks labeled with SW, NW, SE and NE, respectively. This borderline is unique due to the uniqueness of w, s, e, n. Further, the SW- and NE-blocks as well as the SE- and NW-blocks are equally long due to $+1_{p_1}$- and $+1_{p_2}$-closeness of the elements in the border. The formula φ_{center} ensures that all elements within this border are present and φ_{tiling} ensures that the resulting grid is properly tiled. ◀

The straight forward proof of the previous result relies on arbitrarily large equivalence classes of $+1_{p_1}$ and $+1_{p_2}$. However, undecidability already holds already for 2-bounded structures. This gives a tight bound on the extensibility of the decidability of $\mathrm{FO}^2(+1_l, +1_p, \leq_p)$ on k-bounded structures to structures with two preorders.

Theorem 17. *Finite satisfiability of two-variable logic with two additional 2-bounded preorder successor relations is undecidable.*



$\leq_{p_2}$
$\leq_{p_1}$
$\mathcal{T}_{13}^-$ $\mathcal{T}_{13}^+$
$\mathcal{T}_{12}^-$ $\mathcal{T}_{12}^+$
$\mathcal{T}_{11}^-$ $\mathcal{T}_{11}^+$
$\mathcal{T}_{23}^-$ $\mathcal{T}_{23}^+$
$\mathcal{T}_{22}^-$ $\mathcal{T}_{22}^+$
$\mathcal{T}_{21}^-$ $\mathcal{T}_{21}^+$
$\mathcal{T}_{33}^-$ $\mathcal{T}_{33}^+$
$\mathcal{T}_{32}^-$ $\mathcal{T}_{32}^+$
$\mathcal{T}_{31}^-$ $\mathcal{T}_{31}^+$
$\mathcal{T}_{43}^+$ $\mathcal{T}_{43}^-$
$\mathcal{T}_{42}^+$ $\mathcal{T}_{42}^-$
$\mathcal{T}_{41}^+$ $\mathcal{T}_{41}^-$

Figure 6 How the tiling $\mathcal{T} = \mathcal{T}_{11}\mathcal{T}_{12}\mathcal{T}_{13}|\mathcal{T}_{21}\mathcal{T}_{22}\mathcal{T}_{23}|\mathcal{T}_{31}\mathcal{T}_{32}\mathcal{T}_{33}|\mathcal{T}_{41}\mathcal{T}_{42}\mathcal{T}_{43}$ is encoded as a $(+1_{p_1}, +1_{p_2})$-structure with 2-bounded equivalence classes.

Proof sketch. The problem Tiling is reduced to finite satisfiability of $\text{FO}^2(+1_{p_1}, +1_{p_2})$ with 2-bounded equivalence classes. Here, for technical reasons we assume that a valid tiling has at least 3 columns and 4 rows, i.e. $m \geq 3$ and $n \geq 3$.

We say that x is northwest of y if $+1_{p_1}(x, y)$ and $+1_{p_2}(y, x)$. Analogously for southwest, southeast and northeast.

Given a tiling $\mathcal{T}$ of a $m \times n$ grid, we can obtain a $(+1_{p_1}, +1_{p_2})$-structure $\mathcal{M}(\mathcal{T})$ with only two elements per equivalence class as shown in Figure 6.

For the moment, we allow the predicates $\sim_{p_1}$ and $\sim_{p_2}$ and will later describe how to get rid of those. We construct a $\text{FO}^2(+1_{p_1}, +1_{p_2})$-formula $\varphi = \varphi_{label} \wedge \varphi_{row} \wedge \varphi_{next} \wedge \varphi_{start}$ whose models encode valid tilings. The formula φ uses unary predicates from T and from $C = \{C_-, C_+\}$.

The formula φ_{label} ensures that every element carries exactly one label from T and one label from C.

Intuitively, the formula φ_{row} ensures that every row $r = t_1 \ldots t_n$ of a valid tiling is represented by elements $x_1, \ldots, x_n$, $y_1, \ldots, y_n$ such that

- x_i and y_i carry label t_i for every $i \in \{1, \ldots, n\}$.
- x_{i+1} is northeast of x_i for every $i \in \{1, \ldots, n-1\}$. Similarly for $y_1, \ldots, y_n$.
- $x_i \sim_{p_2} y_i$ for all $i \in \{1, \ldots, n\}$.
- $x_i \in C_-$ and $y_i \in C_+$ for all $i \in \{1, \ldots, n\}$.

Therefore φ_{row} expresses the following conditions:

(R1) For any x with label $s \in T$ which is southwest of some x' with label $s \in T$, the right side of s fits to the left side of t. Furthermore, x and x' carry the same label from C.

(R2) For every x there is a y with $x \sim_{p_2} y$ and
  - x and y carry different labels from C.
  - x and y carry the same label from T.

(R3) If there is an x' northeast of x then, for y with $x \sim_{p_2} y$, there is a y' northeast of y. Similarly for x' southwest of x.

The formula $\varphi_{nextline}$ ensures that the encodings of two successive rows are consistent. Therefore $\varphi_{nextline}$ expresses that for every element x

(N1) there is a y with $x \sim_{p_1} y$ and
- x and y carry different labels from C.
- if x carries C_+ and $s \in T$ and y carries C_- and $t \in T$, then the topside of s fits to the bottom side of t.

(N2) if there is an x' northeast of x then, for y with $x \sim_{p_1} y$ there is a y' northeast of y. Similarly for x' southwest of x.

Finally, φ_{start} ensures that the of the smallest element with respect to $+1_{p_2}$ is completely labeled with tiles with color c_1 at the bottom side.

It is clear that, for every valid tiling $\mathcal{T}$, the formula φ is satisfied by $\mathcal{M}(\mathcal{T})$. On the other hand, let $\mathcal{M}$ be a model of φ. We show how to obtain a valid tiling $\mathcal{T}$ from $\mathcal{M}$. Let r_1^- be the element of $\mathcal{M}$ that has no predecessor with respect to $+1_{p_2}$ and carries the label C_-. Note, that this element is uniquely determined. Let $r_2^-, \ldots, r_n^-$ be elements from $\mathcal{M}$ such that r_{i+1}^- is northeast of r_i^- for all $i \in \{1, \ldots, n-1\}$ and n is maximal. Then the first line of $\mathcal{T}$ contains the tiles $t_1, \ldots, t_n$ where $t_i \in T$ is the T-label of r_i^-. By (R1), the horizontal tiling constraints are fulfilled, i.e. the right side of t_i and the left side of t_{i+1} are colored equally. Further, due to (R3), there are elements $r_1^+, \ldots, r_n^+$ such that r_i^- and r_i^+ are in the same equivalence class of $+1_{p_2}$, and r_1^+ and r_n^+ have no elements in southwest and northeast direction, respectively. Moreover r_i^+ and r_i^- carry the same label from T and different labels from C (by (R2)). Condition (N1) guarantees the existence of elements ${r'}_1^-, \ldots, {r'}_n^-$ in $\mathcal{M}$ such that r_i^+ and ${r'}_i^-$ are in the same equivalence class of $+1_{p_1}$ and both carry labels from T that are vertically consistent. Further ${r'}_1^-$ and ${r'}_n^-$ do not have elements in in southwest and northeast direction. By (R1) the horizontal tiling constraints for the T-labels of ${r'}_i^-$ and ${r'}_{i+1}^-$ are satisfied. Thus the T-labels of ${r'}_1^-, \ldots, {r'}_n^-$ can be chosen as the second line of $\mathcal{T}$. Inductively we obtain a valid tiling $\mathcal{T}$.

We shortly describe how to get rid of $\sim_{p_1}$ and $\sim_{p_2}$. It can be ensured that every element u carries profile information about every element v which is southwest or northeast of u. The profile information contains the label of v as well as whether there is an element in southwest or northeast direction of v. The conditions (R2), (R3), (N1) and (N2) can be easily rephrased using this profile information. ◀

We denote the relation ${+1_l}^2$ by $+2_l$. The following corollary slightly improves Theorem 4 in [24].

Corollary . *Finite satisfiability of* $\mathrm{FO}^2(+1_{l_1}, +2_{l_1}, +1_{l_2}, +2_{l_2})$ *is undecidable.*

Proof. We show how to projectively characterize a 2-bounded preorder successor relation $+1_p$ using relations $+1_l$ and $+2_l$. Therefore we construct a sentence $\varphi_{+1_p}(x, y)$ using binary predicates $+1_l$, $+2_l$ and one unary predicate O such that for every binary relation R the following conditions are equivalent:

i) R is a 2-bounded preorder.
ii) There are binary relations $+1_l$ and $+2_l$ and a unary predicate O such that $R = \{(u, v) \mid (u, v) \models \varphi_{+1_p}(x, y)$.

Then the result follows by replacing $+1_{p_1}$ and $+1_{p_2}$ by $\varphi_{+1_{p_1}}(x, y)$ and $\varphi_{+1_{p_2}}(x, y)$, respectively, in the formula φ from the previous proof.

Besides $+1_l$ and $+2_l$, the formula $\varphi_{+1_p}(x, y)$ uses a unary predicate O. The formula is true for (x, y) if the following conditions are satisfied:

- Every other position (with respect to $\leq_l$) is labeled with O, i.e. the first element is not labeled with O, and O-labeled elements alternate with elements not labeled with O.
- If x is labeled with O then $+1_l(x, y)$. If x is not labeled with O then $+2_l(x, y)$.

Now, for a given 2-bounded preorder successor relation defined by

$$\{a_1, a'_1\} \prec \{a_2, a'_2\} \prec \ldots \prec \{a_n, a'_n\}$$

the relations $+1_l$ and $+2_l$ given by

$$a_1 < a'_1 < a_2 < a'_2 < \ldots a_n < a'_n$$

and $O = \{a'_1, \ldots, a'_n\}$ prove 'i $\Rightarrow$ ii'. The other direction is similar. ◀

The following theorems complement results from [3] and [29]. The proofs use similar methods as used in those works.

Theorem 19. *Finite satisfiability of* $\mathrm{FO}^2(+1_l, \leq_l, +1_p)$ *is undecidable.*

Proof. The proof follows the lines of the proof of Proposition 29 in [3].
We reduce from the undecidable problem

Problem:	PCP
Input:	A sequence $(u_1, v_1), \ldots, (u_k, v_k)$, where every $u_i, v_i \in \Sigma^*$.
Question:	Is there a non-empty, finite sequence $\vec{i} = i_1, \ldots, i_m$ such that $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$?

Let $I = (u_1, v_1), \ldots, (u_k, v_k)$ be an instance of PCP. We construct a $\mathrm{FO}^2(\leq_l, +1_l, +1_p)$-sentence φ that has a finite model if and only if I has a solution. The sentence φ uses unary predicates from Σ as well as the two unary predicates U, V, and expresses the following conditions:

(C1) The string projection of $\leq_{l_1}$ is $u_{i_1} v_{i_1} \ldots u_{i_m} v_{i_m}$ for some $m \in \mathbb{N}$. Elements corresponding to some u_i and v_i are marked with U and V, respectively.

(C2) Every equivalence class of $+1_{p_2}$ contains exactly two elements such that
- One is marked with U and one is marked with V.
- Both carry the same label from Σ.

(C3) Positions $x_1, \ldots, x_{|u|}$ corresponding to the positions of $u := u_{i_1} \ldots u_{i_m}$ fulfill $+1_p(i, i+1)$ for all $i \in \{1, \ldots, |u| - 1\}$. Analogously for v.

The first two conditions can be easily expressed in $\mathrm{FO}^2(\leq_l, +1_l, +1_p)$. The third condition can be ensured by the formula

$$\forall x \forall y (U(x) \land U(y) \land x < y \rightarrow \neg{+1_p}(x, y))$$

Now, from a solution $\vec{i} = i_1 \ldots i_m$ a model of φ can be constructed easily. On the other hand, let $\mathcal{M}$ be a a model of φ. By (C1), the string projection of $\mathcal{M}$ is of the form $u_{i_1} v_{i_1} \ldots u_{i_m} v_{i_m}$. The U- and V-labeled elements are ordered with respect to $\leq_p$ due to (C3). Thus, (C2) implies that $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$. ◀

▶ **Theorem C.2.** *Finite satisfiability of* $\mathrm{FO}^2(+1_{p_1}, \leq_{p_2})$*, i.e. two-variable logic with one additional preorder successor relation and one additional preorder relation, is undecidable.*

Proof. The proof follows the lines of the proof of Theorem 4.2 in [29].
◀