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 Mathematical Logic

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

Digital Object Identifier 10.4230/LIPIcs.CSL.2013.484

1 Introduction

The undecidability of the satisfiability and finite satisfiability problem for first-order logic [6, 32, 31] lead to a quest for decidable yet expressive fragments (see for example [3, 15]).

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 [25], in fact they are complete for NExpTime [11]. 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 [26, 1, 19, 20, 18, 28, 30].

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.

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

Data words, introduced in [4], extend usual words by assigning data values to every position. Applications of data words arise for example in verification, where they can be used 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 [1].

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 [33]. In interval temporal logic operators like 'after', 'during' and 'begins' allow for moving along intervals [14]. The connection of intervals to the two-dimensional setting becomes clear when one interprets an interval $[a, b]$ as point (a, b). In [27] 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 [23]. This paper contains a gap (the reduction to Presburger automata is wrong) which can, however, be fixed using the same technique. In [10] 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 [5]. When two linear orders and one of their successors can be accessed the problem is decidable as well [27]. 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 [27]. 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 [27]. Access to a linear order with its successor and either preorder or preorder successor yields undecidability. The former is proved in [2], 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 [2], 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 first author was supported by funding from European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454. 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 $u \leq_p v$.

[2] In this paper all preorders are total.

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 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 $\mathrm{FinOrd}(O)$.

A *word* w over an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is a finite sequence $\tau_1 \ldots \tau_n$ 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 $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ correspond to $+1_l$-structures with unary relations $\mathcal{P} = (P_{\sigma_1}, \ldots, P_{\sigma_k}\}$. 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 $\vec{v}_1 \ldots \vec{v}_l$ of tuples from $\mathbb{N}^{\Sigma}$. A preorder word w can be identified with a preorder $\leq_p$ with Σ-labeled elements where each $\vec{v}_i = (n_{\sigma_1}, \ldots, n_{\sigma_k})$ 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 $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ 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 $w = (\sigma_1, d_1) \ldots (\sigma_n, d_n)$. 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, $C_1 \leq_p \ldots \leq_p C_l$ be the classes of w. The *string projection* of w is the word $\sigma_1 \ldots \sigma_n$ over Σ and is denoted by $sp(w)$. The *preorder projection* $pp(w)$ is the preorder word that corresponds to $\leq_p$, that is $pp(w) = \vec{c}_1 \ldots \vec{c}_l$ where each $\vec{c}_i = (n_{\sigma_1}, \ldots, n_{\sigma_k})$ 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.

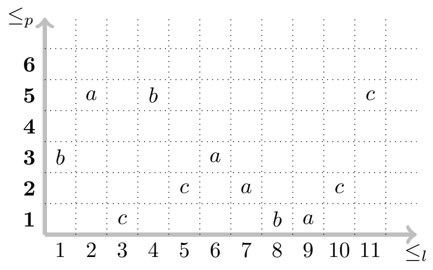

Figure 1 The ordered structure representing the ordered data word $(b,\mathbf{3})(a,\mathbf{5})(c,\mathbf{1})(b,\mathbf{5})(c,\mathbf{2})(a,\mathbf{3})(a,\mathbf{2})(b,\mathbf{1})(a,\mathbf{1})(c,\mathbf{2})(c,\mathbf{5})$. The classes are $\{3,8,9\} \leq_p \{5,7,10\} \leq_p \{1,6\} \leq_p \{2,4,11\}$, 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.

Denote by 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 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 EMSO$(+1_l)$-formulas. Similarly preorder words and ordered data words are interpretations for EMSO$(+1_p, \leq_p)$- and EMSO$(+1_l, +1_p, \leq_p)$-formulas, respectively.

The language $L(\varphi)$ of $\varphi \in$ 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 EMSO$(+1_l, \leq_l)$ is equivalent to finite state automata. This holds even for EMSO$^2(+1_l)$. In the next section we introduce an automaton model which is equivalent to 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 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 [2]. In the next section ordered data automata are shown to be equivalent to 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 $\vec{w}_1 \ldots \vec{w}_n$ is a sequence of transitions $\delta_1 \ldots \delta_n$ 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 $w = (\sigma_1, d_1) \ldots (\sigma_n, d_n)$ 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 $C_1 \leq_p \ldots \leq_p C_r$ 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 $(\sigma_1, m(1)) \ldots (\sigma_n, m(n))$ 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 [23].

▶ **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. [12] 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

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

▶ **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 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 $\delta_=(x,y) = \perp$, 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 $\mathrm{EMSO}^2(+1_l, +1_p, \leq_p)$ we can conclude that the result from the previous section carry over to the case of k-bounded ordered data words, 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 $\mathrm{EMSO}^2(+1_l, +1_p, \leq_p)$ formula φ.

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

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

▶ **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 [23], where the finite satisfiability problem of $\mathrm{FO}^2(+1_{l_1}, +1_{l_2})$ 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 [24, 21]. 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

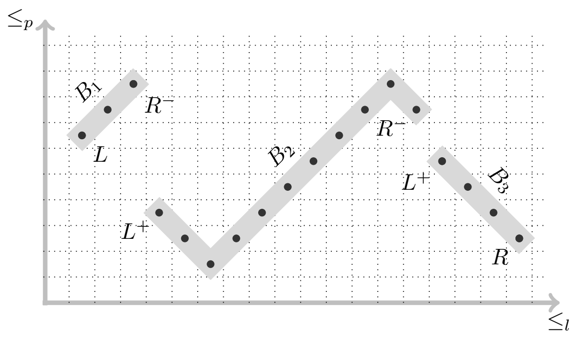

Figure 2 Blocks in the ordered-structure-representation of a 3-bounded ordered data word w. Each • represents one element of w, the $\le_l$-axis represents positions whereas the $\le_p$-axis represents data values. Labels are omitted for clarity.

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 $w' = \vec{w}'_1 \dots \vec{w}'_m$ 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.

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 $\le_p$-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 $\{1, \dots, 2k\}$ such that $N(B) \neq N(B')$ if B and B' are $\le_l$-adjacent blocks or $\le_p$-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 $\Sigma_B = \{1, \dots, 2k\} \times (\{L, L^+, L^-, C\} \times \{R, R^+, R^-, C\})$. 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 $\Sigma_R = \Sigma' \times Q^2_\mathcal{B} \times Q^2_\mathcal{C} \times Q^2_\mathcal{B}$ where $r_\mathcal{B}$, $r_\mathcal{C}$ and $b_\mathcal{B}$ 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 $w = w_1 \dots w_n$ 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 $\mathsf{annpp}(w, \rho)$.

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

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 $s_{i+1} \ll_p t_i$.

[4] Recall that $\vec{w}'_{s_i}$ can be identified with the equivalence class of the preorder corresponding to B'_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 $\{0, \ldots, k\}^{|\Sigma|}$.

▶ **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).*

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). ◀

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.*

▶ **Corollary 16.** *Finite satisfiability of* $\text{FO}^2(+1_l, +1_p, \leq_p)$ *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 [23].

▶ **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 [2] and [28]. 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.

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 [22], 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 $\mathrm{FO}^2(+1_{l_1}, \dots, +1_{l_m})$ is undecidable. A method for proving undecidability of $\mathrm{FO}^2(+1_{l_1}, \dots, +1_{l_m})$ should not extend to $\mathrm{FO}^2(F_1, \dots, F_m)$ where $F_1, \dots, F_m$ 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 $\mathrm{FO}^2(F_1, \dots, F_m)$ is decidable since one can express that some arbitrary interpreted binary predicate R is a permutation by using two-variable logic with counting quantifiers which in turn is decidable by [13]. This is an observation by Juha Kontinen.

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

Temporal logics on data words have seen much research recently [7, 8, 16]. 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 [29] 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?*

[5] Under elementary reductions.

Logic	**Complexity** (lower/upper)	**Comments**
One linear order		
$\mathrm{FO}^2(+1_l)$	NExpTime-complete	[9]
$\mathrm{FO}^2(\leq_l)$	NExpTime-complete	[26, 9]
$\mathrm{FO}^2(+1_l, \leq_l)$	NExpTime-complete	[9]
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	[28]
Two linear orders		
$\mathrm{FO}^2(+1_{l_1}, +1_{l_2})$	NExpTime-complete	[23, 10, 5]
$\mathrm{FO}^2(+1_{l_1}, \leq_{l_2})$	NExpTime/ExpSpace	[28]
$\mathrm{FO}^2(+1_{l_1}, +1_{l_2}, \leq_{l_2})$	Multicounter-Emptiness[5]	★, Corollary 11 and Theorem 15
$\mathrm{FO}^2(+1_{l_1}, \leq_{l_1}, \leq_{l_2})$	NExpTime/ExpSpace	[28]
$\mathrm{FO}^2(+1_{l_1}, \leq_{l_1}, +1_{l_2}, \leq_{l_2})$	Undecidable	[23]
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 20
$\mathrm{FO}^2(\leq_{p_1}, \leq_{p_2})$	Undecidable	[27]
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	[2]
$\mathrm{FO}^2(+1_l, +1_p, \leq_p)$	? (see discussion)	★, Special case: Theorem 10
$\mathrm{FO}^2(\leq_l, +1_p, \leq_p)$	ExpSpace-complete	[28]
Many orders		
$\mathrm{FO}^2(\leq_{l_1}, \leq_{l_2}, \leq_{l_3})$	Undecidable	[17]
$\mathrm{FO}^2(+1_{l_1}, +1_{l_2}, +1_{l_3})$	?	
$\mathrm{FO}^2(+1_{l_1}, +1_{l_2}, +1_{l_3}, \ldots)$	?	

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 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 [28].

References

1 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.

2 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.

3 Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem.* Springer Verlag, 2001.

4 Patricia Bouyer. A logical characterization of data languages. *Inf. Process. Lett.*, 84(2):75–85, 2002.

5 Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and trees. In *LICS*, 2013 (To appear).

6 Alonzo Church. A note on the Entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41, 1936.

7 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.

8 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

9 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.

10 Diego Figueira. Satisfiability for two-variable logic with two successor relations on finite linear orders. *CoRR*, abs/1204.2495, 2012.

11 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.

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

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

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

15 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.

16 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.

17 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.

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

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

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

21 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.

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

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

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

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

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

27 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.

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

29 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.

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

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

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

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