

Two-Variable Logic with a Linear Successor and a Preorder

Amaldev Manuel¹ and Thomas Zeume²

¹ Institute of Mathematical Sciences, Chennai
amal@imsc.res.in

² Technische Universität Dortmund
thomas.zeume@cs.uni-dortmund.de

Abstract

This paper studies the finite satisfiability problem of first order logic restricted to two variables (FO^2) on structures with a linear order \leq_l , a total preorder \leq_p , their induced successor relations $+1_l$ and $+1_p$ as well as an arbitrary number of unary relations. We prove that the finite satisfiability of $\text{FO}^2(+1_l; +1_p, \leq_p)$ on structures, where the size of the equivalence classes of \leq_p is bounded by some fixed k , is decidable. As a corollary, two-variable logic is decidable on structures with two linear order successors and an order corresponding to one of the successors. Moreover it is shown that both problems are at least as hard as the reachability problem for vector addition systems. While the decidability of $\text{FO}^2(+1_l; +1_p, \leq_p)$ on general structures remains open, we prove undecidability for some other extensions. This line of research is in the context of classical logic over data words.

1998 ACM Subject Classification F.4 MATHEMATICAL LOGIC AND FORMAL LANGUAGES

Keywords and phrases Two-variable logic, Data words, Automata on data words.

1 Introduction

The two-variable fragment of first order logic (two-variable logic or FO^2 for short) is known to be reasonably expressive for many purposes. Yet the satisfiability problem is decidable in NEXPTIME [13, 5]. As transitivity cannot be expressed in two-variable logic, many extensions of two-variable logic by additional relations that are interpreted as equivalence classes or orders have been considered [14, 10, 11].

Extending two-variable logic by n additional order relations can be seen as using two-variable logic on n -dimensional structures. In this work we continue the examination of the two-dimensional setting, that was started in [2, 12, 15].

The motivation for looking at the two-dimensional setting is so-called data words. A *data word* is a word, that is, a finite sequence of symbols from a finite alphabet, where every position also carries a value from a possibly infinite domain. An interest in data words and data trees arises from applications in database theory, where XML documents can be modeled by data trees in which the symbols correspond to the tags and the data values to text or attribute values. On the other hand, (infinite) data words can also be considered as traces of computations in a distributed environment, where symbols correspond to states of processes and data values encode process numbers. Recently many logics and automata models have been considered for data words and data trees (see [17] for a gentle introduction).

Data words with a linearly ordered data domain can be seen as finite structures with two order relations. More precisely, data words can be represented by

- unary relations denoted by Σ to encode symbols at positions,
- a linear order on the positions denoted by \leq_l ,



© Amaldev Manuel and Thomas Zeume;

licensed under Creative Commons License ND

Leibniz International Proceedings in Informatics

LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ a preorder on the positions denoted by \leq_p induced by the linear order of the data domain. 1

For finite structures, the linear order and the preorder induce successor relations. This is the setting we are interested in. 2
3

Related work. There are already some partial results. When only a linear order and a preorder predicate are available, then finite satisfiability is decidable and complete for EXPSpace [15]. It even remains decidable when the successor predicate of the preorder is added [16], however, it is undecidable if the successor predicate of the linear order is added [2]. The undecidability proof from [2] can be adapted to the case where linear order, linear order successor and preorder successor predicates are available. In [12] it was shown that if two linear orders (for data words this means that every data value occurs at most once) and their successor predicates are available then finite satisfiability is undecidable. The problem becomes decidable when only the successor relations are allowed. 4
5
6
7
8
9
10
11
12

The main question for this paper is, whether the decidability result from [12] can be extended to a linear order, a preorder and its corresponding successor, $\text{FO}^2(+1_l; +1_p, \leq_p)$ for short. For our proof, we adapt a version of data automata, introduced in [2], to ordered data words. 13
14
15
16

Another automaton model for data words which got some attention is register automaton [8, 4]. Very recently, in [18] the authors generalized this automaton model to ordered data words both finite and infinite. They also introduce a temporal logic for ordered data words. Yet this logic is incomparable to the logics considered in this paper. 17
18
19
20

Results. We are not able to settle the most general setting that is, whether finite satisfiability of $\text{FO}^2(+1_l; +1_p, \leq_p)$ is decidable. However, when the preorder is k -bounded, that is, there are at most k equivalent elements with respect to the preorder, then we obtain decidability of the finite satisfiability problem. As two-variable logic can express that preorder equivalence classes are of size at most 1, this result implies decidability of two-variable logic with two additional linear orders and one of their corresponding successors. In the setting of data words, k -boundedness means that a data value can occur in at most k -positions. Note, that the undecidability results for ordered data words and data words with a successor relation already hold if data values are allowed to appear at most twice. Therefore decidability for $\text{FO}^2(+1_l; +1_p, \leq_p)$ for k -bounded structures is a strong indication of decidability of the general problem. 21
22
23
24
25
26
27
28
29
30
31

The proof outline is classical. First, we introduce an automaton model that is suited for our setting, called ordered data automaton. Ordered data automaton – similarly to data automata from [2] – does a two-stepped run on structures. In the first step, a transducer runs over the structure in the direction indicated by the linear order induced by $+1_l$. Then a second automaton runs on the resulting structure in the direction of the preorder \leq_p . We prove that ordered data automata are expressively equivalent to $\text{FO}^2(+1_l; +1_p, \leq_p)$ on k -bounded structures. Secondly, the emptiness problem for ordered data automata is reduced to the emptiness problem of multicounter automata in a construction similar, but much more involved, to the construction in [12]. 32
33
34
35
36
37
38
39
40

Moreover we prove that satisfiability of $\text{FO}^2(+1_l; +1_p, \leq_p)$ is at least as hard as the reachability problem for vector addition systems and undecidability of some further variants. 41
42

Organization. After some basic definitions in Section 2, we introduce ordered data automata in Section 3.1 and prove that they are expressively equivalent to $\text{FO}^2(+1_l; +1_p, \leq_p)$ on k -bounded structures. Section 3.2 is devoted to proving decidability of the emptiness 43
44
45

problem for ordered data automata. In Section 4 lower bounds for several variants are proved. We conclude in Section 5.

Acknowledgements. We thank Thomas Schwentick for driving our attention to FO^2 on ordered structures and many helpful discussions.

2 Preliminaries

We use a setup similar to [16] for ordered structures. First the notation for general ordered structures is fixed, afterwards some notions for the two-dimensional case are introduced.

Ordered Structures. We first fix our notation concerning order relations. A *total preorder* \leq_p is a reflexive, transitive and total relation, that is, for every elements u, v, w of the structure (i) $u \leq_p v$ and $v \leq_p w$ implies $u \leq_p w$ and (ii) $u \leq_p v$ or $v \leq_p u$ holds. A *linear order* \leq_l 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 that for two distinct elements u and v both $u \leq_p v$ and $v \leq_p u$ hold. We call two such elements *equivalent with respect to \leq_p* . Thus, a total preorder can be viewed as an equivalence relation \sim_p whose equivalence classes are linearly ordered by $<_p$. Clearly, every linear order is a total preorder with equivalence classes of size one. For any element u , the \sim_p -class of u is denoted by $[u]_{\sim_p}$ (or $[u]$ if \sim_p is clear from the context). The set of all equivalence classes of \sim_p is denoted by S/\sim_p , where S is the universe of the structure. A total preorder is *k-bounded*, if every equivalence class contains at most k elements.

Only finite structures are considered. Therefore, the linear order on the equivalence classes of a total preorder induces a successor relation of the equivalence classes. We write $+1_p(u, v)$ if the equivalence class of v with respect to \leq_p is the successor of the equivalence class of u and we call $+1_p$ the *induced successor relation* of \leq_p . Further we say u and v are $+1_p$ -close, if either $+1_p(u, v)$ or $u \sim_p v$ or $+1_p(v, u)$. Similarly for $+1_l$ -close.

We use binary relation symbols $\leq_l, \leq_{l_1}, \leq_{l_2}, \dots$ that are always interpreted as linear orders, binary relation symbols $\leq_p, \leq_{p_1}, \leq_{p_2}, \dots$ that are interpreted as total preorders, and binary relation symbols $+1_p, +1_{p_1}, +1_{p_2}, \dots$ as well as $+1_l, +1_{l_1}, +1_{l_2}, \dots$ that are interpreted as successor relations of preorders and linear orders, respectively. By \sim_p we denote the underlying equivalence relation of a total preorder \leq_p and by $<_p$ its strict total preorder. Thus, $u \sim_p v$ if and only if $u \leq_p v$ and $v \leq_p u$. And $u <_p v$ if and only if $u \leq_p v$ and $u \not\sim_p v$. We note that \sim_p and $<_p$ can be expressed in two-variable logic, given \leq_p .

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. We always allow an unlimited number of unary relations and specify the numbers of allowed linear orders and total preorders explicitly. For example, a $(+1_{l_1}; +1_{p_2}, \leq_{p_2})$ -structure is a structure with one linear order and one total preorder together with its induced successor relation. We write $(+1_l; +1_p, \leq_p)$ instead of $(+1_{l_1}; +1_{p_2}, \leq_{p_2})$ if no ambiguities arise.

A Linear Successor, a Preorder and Two Variables. Ordered structures with two orders – total preorders or linear orders – have a natural representation as sets of labeled points in the two-dimensional plane, see Figure 1 for an example.

In the following, we introduce some vocabulary for $(+1_l; +1_p, \leq_p)$ -structures. These structures are also called *ordered data words*. An ordered data word is *k-bounded* if \leq_p is *k-bounded*. Let \mathcal{S} be a *k-bounded* $(+1_l; +1_p, \leq_p)$ -structure over universe S . The *linear*

order projection (\leq_l -projection) $\text{lp}(\mathcal{S})$ of \mathcal{S} is the restriction of \mathcal{S} to unary relations as well as the $+1_l$ -relation, i.e. $\text{lp}(\mathcal{S}) = (S, \Sigma, +1_l)$. The \leq_l -projection is identified with the sequence of the labels of all elements in linear order. For example, the \leq_l -projection of the $(+1_l; +1_p, \leq_p)$ -structure from Figure 1 is d, c, b, a, b, d .

Slightly abusing the notation we use $+1_p$ and \leq_p to denote the successor relation and linear order relation on S/\sim_p that are induced by \leq_p . Let $\text{parikh}_k(\Sigma)$ be the set of parikh vectors over Σ whose sum of components is at most k . Every $[a] \in S/\sim_p$ can be labeled by a symbol p from $\text{parikh}_k(\Sigma)$ such that, for every $\sigma \in \Sigma$, p indicates the number of σ -labeled elements in $[a]$. The preorder projection (\leq_p -projection) of \mathcal{S} is $\text{pp}(\mathcal{S}) = (S/\sim_p, \text{parikh}_k(\Sigma), +1_p, \leq_p)$, i.e. the \leq_p -projection of \mathcal{S} considers \sim_p -equivalence classes as single elements. We will identify the preorder projection with the sequence p_1, \dots, p_m where $p_i \in \text{parikh}_k(\Sigma)$ is the parikh image of the i th element of S/\sim_p with respect to \leq_p . Thus the preorder projection of the $(+1_l; +1_p, \leq_p)$ -structure from Figure 1 is $(0, 1, 0, 0), (1, 0, 0, 0), (0, 0, 1, 0), (0, 1, 0, 1), (0, 0, 0, 1)$ where, for instance, the second component of all vectors represents the number of occurrences of the label b .

A (binary) type is a maximal consistent conjunction of binary literals using $+1_l, +1_p, \leq_p$ and $=$ as well as unary literals. A preorder type is a type that does not use $+1_l$ and unary literals, similarly a linear order type is a type that does not use $+1_p, \leq_p$ and unary literals. We denote the set of preorder and linear order types by Γ_p and Γ_l , respectively. A pair (u, v) is of preorder type $\tau_p \in \Gamma_p$, if $(u, v) \models \tau_p$, analogously for linear order types. Denote the preorder type of (u, v) by $\tau_p(u, v)$. Analogously for linear order types. Occasionally we will abbreviate the preorder type $\tau(x, y)$ of a pair (u, v) satisfying $+1_p(u, v)$ by $+1$, the preorder type of a pair (u, v) satisfying $\neg +1_p(u, v) \wedge u \leq_p v \wedge u \neq v$ by $+\infty$, similarly the other preorder types are abbreviated by $0, -1, -\infty$. Analogously linear order types are abbreviated by $+1, 0, -1$ and ∞ . Note that linear order types cannot distinguish between $+\infty$ and $-\infty$ (as only $+1_l$ is available).

Two-variable logic is the restriction of predicate logic to formulas that only use (at most) two variables x and y . We denote two-variable logic by FO^2 and two-variable logic on ordered structures of signature C by $\text{FO}^2(C)$.

3 One Linear Successor and a Preorder

In this section we prove the main result of the paper.

► **Theorem 1.** *The finite satisfiability problem for $\text{FO}^2(+1_l; +1_p, \leq_p)$ on k -bounded structures is decidable.*

As 1-boundedness can be axiomatized in FO^2 by $\forall x \forall y (x \neq y \rightarrow \neg x \sim_p y)$, the following is an immediate corollary.

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

Thus the main result generalizes Theorem 3 from [12], where the finite satisfiability problem of $\text{FO}^2(+1_{l_1}; +1_{l_2})$ was shown to be decidable. The extension to decidability for k -bounded structures is a strong indicator that the finite satisfiability problem for unbounded structures is decidable as well since, for example, $\text{FO}^2(+1_l, \leq_l; \leq_p)$ and $\text{FO}^2(+1_{p_1}; +1_{p_2})$ are undecidable already when the class of structures is restricted to 2-bounded structures, see [2] and Proposition 7.

In the first part of the section we define an automaton model suited for $\text{FO}^2(+1_l; +1_p, \leq_p)$ on structures where \leq_p is k -bounded. Therefore we extend data automata presented in [2] to

k -ordered data automata (k -ODA) and prove that $\text{FO}^2(+1_l; +1_p, \leq_p)$ on k -bounded structures and k -ODA are expressively equivalent. By reducing the emptiness problem for k -ODA to the emptiness problem for multicounter automata, decidability of the finite satisfiability problem for $\text{FO}^2(+1_l; +1_p, \leq_p)$ is obtained.

3.1 Automata over Ordered Data Words

We introduce k -ordered data automata. Roughly speaking, an ordered data automaton $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ consists of a non-deterministic finite state transducer \mathcal{B} and a finite state automaton \mathcal{C} . Intuitively the transducer \mathcal{B} is running on an input structure with respect to the linear order induced by $+1_l$. The automaton \mathcal{C} runs on the result of \mathcal{B} with respect to \leq_p . By showing basic closure properties and expressive power of k -ODA, we prepare for proving the main result in this section, namely that $\text{FO}^2(+1_l; +1_p, \leq_p)$ on k -bounded structures and k -ODA are expressively equivalent.

Before defining k -ordered data automata, we fix some additional notations. For the rest of this section let \mathcal{S} be a k -bounded $(+1_l; +1_p, \leq_p)$ -structure over universe S for some fixed k , let the linear order induced by $+1_l$ be denoted by \leq_l and let \mathcal{P} be the set of unary relations from \mathcal{S} . Every element in S can be seen as carrying exactly one label from $\Sigma := 2^{\mathcal{P}}$.

The \leq_p -marking of \mathcal{S} is obtained as follows. Every element $v \in S$ with $+1_l$ -predecessor u and $+1_l$ -successor w is labeled with $(\tau_p(v, u), \tau_p(v, w))$ that is with the preorder types of (v, u) and (v, w) . If v has no predecessor, it is labeled with $(-, \tau_p(v, w))$, similarly if v has no successor. Denote the \leq_p -marking of \mathcal{S} by $\text{pm}(\mathcal{S})$. Thus, in $\text{pm}(\mathcal{S})$ every element v knows where its $+1_l$ -predecessor and successor occur with respect to the preorder.

We propose a variant of Data automata on k -class bounded ordered data words. A k -bounded Ordered Data Automaton (k -ODA) is a composite automaton $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ where \mathcal{B} is a non-deterministic finite state transducer with an input alphabet $\Sigma \times (\Gamma_p \cup \{-\})^2$ and an output alphabet Π . The automaton \mathcal{C} is a finite state automaton working on words over the alphabet $\text{parikh}_k(\Pi)$. Intuitively the transducer \mathcal{B} is running over \mathcal{S} with respect to the linear order induced by $+1_l$. The automaton \mathcal{C} runs on the result of \mathcal{B} with respect to \leq_p .

Given a $(+1_l; +1_p, \leq_p)$ -structure \mathcal{S}' a k -ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ works on $\mathcal{S} = \text{pm}(\mathcal{S}')$ as follows. The transducer \mathcal{B} works on the linear order projection of \mathcal{S} yielding a run ρ_B which in turn defines the unique (for each run) new structure \mathcal{S}' . The automaton \mathcal{C} runs over the preorder projection of the structure \mathcal{S}' yielding a run ρ_C . Automaton \mathcal{A} accepts the structure \mathcal{S}' if both ρ_B and ρ_C are accepting. The set of all k -bounded structures accepted by \mathcal{A} is denoted by $L(\mathcal{A})$. The transducer \mathcal{B} is called *linear order automaton*, the automaton \mathcal{C} is called *preorder automaton*.

The following example will be used later.

► **Example 3.** Consider the set L of k -bounded structures \mathcal{S} with the following two properties:

i) Only one element $v \in S$ is labeled with $b \in \Sigma$.

ii) $\tau_l(u, v) = \infty$ and $\tau_p(u, v) = +\infty$ for every a -labeled element $u \in S$.

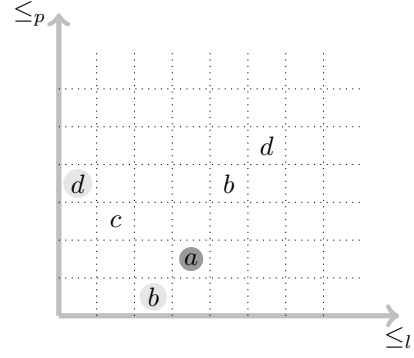
A k -ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ can recognize L as follows. The transducer \mathcal{B} verifies that there is only one element v labeled with b ; and that v has no $+1_l$ -close element labeled with a . \mathcal{C} verifies that, with respect to the preorder, no a -labeled element follows after the b -labeled element. Further, \mathcal{C} verifies that $[u]$ with $+1_p([u], [v])$ contains no a -labeled element.

If condition ii) is replaced by

ii') $\tau_l(u, v) = \infty$ and $\tau_p(u, v) = +1$ for every a -labeled element $u \in S$.

the finite state automaton \mathcal{C} checks that $+1_p(u, v)$ for all a -labeled elements u .

■ **Figure 1** A $(+1_l; +1_p, \leq_p)$ -structure and profile information. Columns are ordered by \leq_l , rows are ordered \leq_p , i.e. every box represents the intersection of one \leq_p -class and one \leq_l -class. The profile of the a -labeled element contains, among others, the types $\tau_1 = d(y) \wedge \neg +1_l(x, y) \wedge \neg +1_l(y, x) \wedge x \leq_p y \wedge \neg +1_p(x, y) \wedge x \neq y$ and $\tau_2 = b(y) \wedge +1_l(y, x) \wedge +1_p(y, x)$.



Closure properties and expressive power. Firstly, we examine some properties of k -ODA. The following lemmata can be proved straight forwardly. 1
2

► **Lemma 4.** *There exists k -ODA \mathcal{A}_\emptyset and $\mathcal{A}_{\bar{\emptyset}}$ which accept no k -bounded $(+1_l; +1_p, \leq_p)$ -structure and all k -bounded $(+1_l; +1_p, \leq_p)$ -structures, respectively. 3
4*

► **Lemma 5.** *Languages accepted by k -ODA are closed under union, intersection and renaming. 5
6*

The following proposition can be proved like Lemma 3 in [12]. 7

► **Proposition 1.** *Languages accepted by k -ODA are not closed under complementation. 8*

In preparation of the equivalence proof, we introduce the notion of the profile of an element of a k -bounded $(+1_l; +1_p, \leq_p)$ -structure \mathcal{S} . The *profile* $\text{pro}(u)$ of an element $u \in \mathcal{S}$ contains exactly those binary types $\tau(x, y)$ such that there is a $v \in \mathcal{S}$ with $\tau(u, v)$. See Figure 1 for an example. The profile $\text{pro}(\mathcal{S})$ of \mathcal{S} is obtained by labeling every element of \mathcal{S} with its profile. 9
10
11
12
13

Now we give a first demonstration of the expressive power of k -ODA by showing that k -ODA can guess and verify profile information of all elements of an input structure. 14
15

► **Lemma 6.** *There is a k -ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ such that \mathcal{A} accepts a k -bounded structure \mathcal{S} if and only if \mathcal{B} outputs $\text{pro}(\mathcal{S})$, i.e. there is a k -ODA that can guess and verify the profile of a structure. 16
17
18*

Proof. Let $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ be a k -ODA such that \mathcal{B} guesses profiles for every element and also forwards the preorder marking to the automaton \mathcal{C} . 19
20

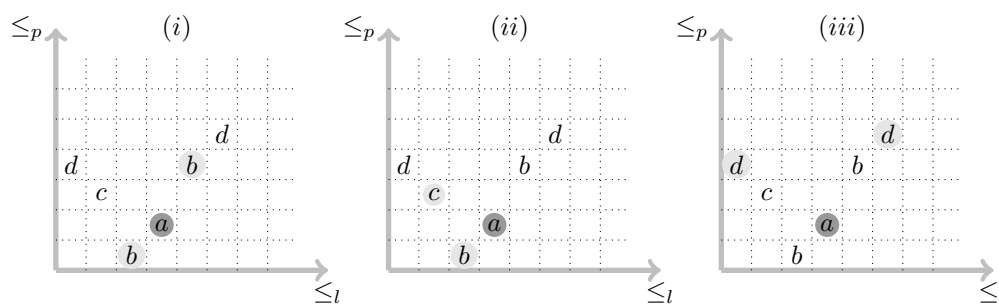
We show how \mathcal{B} and \mathcal{C} can, for every type τ and all elements u , verify that 21

- if u is labeled with τ , then there is a v such that (u, v) is of type τ . 22
- if u is not labeled with τ , then there is no v such that (u, v) is of type τ . 23

We fix a type $\tau(x, y)$ and assume that τ requires the unary types of x and y to be σ and σ' , respectively. Basically, by labelling u with $\tau(x, y)$, \mathcal{B} claims that there is an element v labeled with σ' in a certain direction from u (which is specified by the binary literals in $\tau(x, y)$). Let $\tau_l \subseteq \tau(x, y)$ (resp. $\tau_p(x, y) \subseteq \tau(x, y)$) be maximal with $\tau_l \subseteq \Gamma_l$ (resp. $\tau_p \subseteq \Gamma_p$). 24
25
26
27

Now, we show how the following main cases can be verified: 28

- i) $|\tau_l(x, y)| \leq 1$ 29
- ii) $|\tau_p(x, y)| \leq 1$ 30
- iii) $\tau_l = \infty$ and $|\tau_p(x, y)| = \infty$ 31



■ **Figure 2** Case distinction of binary types for Lemma 6. Elements highlighted light gray are relevant for verifying the profile of the a -labeled element for the respective case.

See Figure 2 for a graphical illustration of these cases. The cases i) and ii) can be found in the Appendix.

For iii) assume $\tau_l = \infty$ and $\tau_p(x, y) = +\infty$. Firstly, we observe the following. There is some bound $m \in \mathbb{N}$ such that any σ -position u with at least m later occurrences of σ' (with respect to the preorder projection) has profile τ . This follows from the fact that there only boundedly many elements that are $+1_l$ - or $+1_p$ -close to u . Now, \mathcal{B} guesses the last m (with respect to \leq_p) σ' -positions v_1, \dots, v_m and colors them with new labels $\sigma'_1, \dots, \sigma'_m$ (if there are less σ' -positions, \mathcal{B} colors them distinctly). Further, \mathcal{B} colors every position u with tuples $(\tau_1, \sigma_1), \dots, (\tau_m, \sigma_m)$ such that $\tau_i = \tau(u, v_i)$. \mathcal{B} and \mathcal{C} can verify this coloring since $\{v_1, \dots, v_m\}$ is finite (see Example 3).

Now, checking that \mathcal{B} guessed τ correctly for all positions is easy. Let u be an arbitrary position. If u is labeled with τ , then \mathcal{B} checks that u is labeled with some τ_i , $i \in \{1, \dots, m\}$ with $\tau = \tau_i$ (then v_i is a witness with $\tau(u, v_i)$). If u is not labeled with τ then \mathcal{C} verifies that there are at most m σ' -positions after u (with respect to \leq_p) and that $\tau_i \neq \tau$ for all τ_i . Similarly for the other cases of iii). This completes the proof. ◀

In the following we assume that every element of an input structure for a k -ODA \mathcal{A} is labeled with its profile.

Equivalence of k -ODA and $\text{FO}^2(+1_{l_1}; +1_{p_2}, \leq_{p_2})$ on k -bounded structures. Given a formula $\varphi \in \text{EMSO}^2(+1_l; +1_p, \leq_p)$ we define $L_k(\varphi)$ as the set of all k -bounded $(+1_l; +1_p, \leq_p)$ -models of φ . Encoding a k -ODA into a formula can be done in the usual way. The proof of the following proposition can be found in the Appendix.

► **Proposition 2.** *Given a k -ODA \mathcal{A} there is a sentence $\varphi_{\mathcal{A}} \in \text{EMSO}^2(+1_l; +1_p, \leq_p)$ such that $L(\mathcal{A}) = L_k(\varphi_{\mathcal{A}})$.*

Now we show, how to translate a sentence φ into a k -ODA. As a first step φ is transformed into Scott Normal Form

$$\exists X_1 \dots X_n (\forall x \forall y \psi \wedge \bigwedge_i \forall x \exists y \chi_i)$$

where X_1, \dots, X_n are fresh second order variables and ψ, χ_i are quantifier-free formulas (see e.g. [6] for the translation). Since k -ODA are closed under union, intersection and renaming it is sufficient to show that there exist k -ODAs accepting models of formulas of the form $\forall x \forall y \psi$ and $\forall x \exists y \chi$.

► **Lemma 7.** *For every sentence φ of the form $\forall x\forall y \psi$ with ψ quantifier-free there is a k -ODA accepting exactly the k -bounded models of φ .*

Proof. It is straight forward to convert φ into a formula of the form

$$\bigwedge_{\sigma, \tau \in \mathcal{U}} \forall x\forall y(\sigma(x) \wedge \tau(y) \rightarrow \gamma_{\sigma, \tau}(x, y))$$

where $\gamma_{\sigma, \tau}$ are disjunctions of binary types and \mathcal{U} denotes the set of unary types. Again, every conjunct can be checked separately due to closure under intersection. A single condition $\forall x\forall y(\sigma(x) \wedge \tau(y) \rightarrow \gamma(x, y))$ allows for elements a and b with labels σ and τ to be related only in ways that are allowed by γ . That is, certain binary types – namely those that do not occur in the disjunction – are forbidden for a and b . This can be checked by a k -ODA using the information contained in the profiles. For example, if $+1_l(x, y) \wedge x \leq_p y \wedge \neg +1_p(x, y)$ is forbidden by γ , then a k -ODA has to check that, for every element labeled with σ , the type $+1_l(x, y) \wedge x \leq_p y \wedge \neg +1_p(x, y) \wedge \sigma(x) \wedge \tau(y)$ is not contained in the profile. ◀

The following proposition can be proved similarly to the previous one, for details see the Appendix.

► **Lemma 8.** *For every sentence φ of the form $\forall x\exists y \chi$ with χ quantifier-free there is a k -ODA accepting exactly the k -bounded models of φ .*

Proposition 2, Lemma 7 and Lemma 8 immediately give the following theorem, which in turn shows that satisfiability of $\text{FO}^2(+1_l; +1_p, \leq_p)$ over k -bounded structures reduces to the non-emptiness problem of k -ODA.

► **Proposition 3.** *For a language L of k -bounded $(+1_l; +1_p, \leq_p)$ -structures the following are equivalent:*

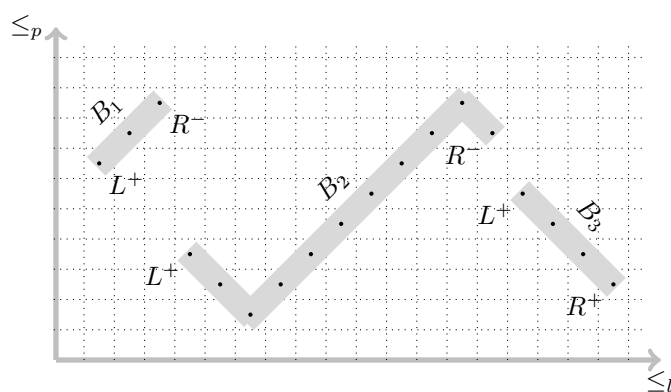
- *There is a k -ODA \mathcal{A} with $L = L(\mathcal{A})$.*
- *There is a sentence $\varphi \in \text{EMSO}^2(+1_l; +1_p, \leq_p)$ such that $L = L_k(\varphi)$.*

3.2 Deciding the Emptiness Problem for k -ODA

Now, we prove the decidability of the emptiness problem of k -ODA by a reduction to the emptiness problem of multicounter automata. Roughly speaking, the run of a k -ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ on an input structure \mathcal{S} will be simulated by a multicounter automaton that reads the preorder projection of \mathcal{S} , guesses and verifies a run of \mathcal{C} , and, meanwhile, builds up a run of \mathcal{B} in the counters. The intricate part of reconstructing a run of \mathcal{B} is that \mathcal{B} sees the \leq_p -marking. The proof will proceed as follows. Lemma 9 deals with those parts of runs of \mathcal{B} that process maximal sequences of consecutive positions where two succeeding positions with respect to $+1_l$ are \leq_p -close. Later, those sequences will be called *blocks*. Constructing a complete run of \mathcal{B} from different blocks will mainly be done in Proposition 4, for the 1-bounded case, and in Theorem 10, for the k -bounded case. The former introduces techniques that will also be used by the latter.

A *multicounter automaton* is, essentially, a finite state automaton equipped with a finite set of counters that can be incremented and decremented. See [1] for a precise definition.

A sequence u_1, \dots, u_n over S is called a *partial block* if, for all i in $\{1, \dots, n-1\}$, u_{i+1} is the $+1_l$ -successor of u_i and u_i and u_{i+1} are close with respect to the preorder. A *block* B is a partial block u_1, \dots, u_n such that elements u_0 and u_{n+1} with $+1_l(u_0, u_1)$ and $+1_l(u_n, u_{n+1})$ (in case they exist) are not \leq_p -close to u_1 and u_n , respectively. The elements u_1 and u_n are called *leftmost* and *rightmost* positions of B . From now on, we assume that u_1 is labeled



■ **Figure 3** Blocks in a snippet of a 3-bounded $(+1_l; +1_p, \leq_p)$ -structure.

with $L^+(B)$ if $u_1 \leq_p u_0$ (and with L if there is no u_0). Similarly for $L^-(B), R^+(B), R^-(B)$ and R . From now on we will assume that u_1 and u_n are labeled by the appropriate L^+ or L^- and R^+ or R^- . The preorder projection $\text{pp}(B)$ of a block B is the preorder projection of S restricted to B . Likewise for the linear order projection. As before, we identify the preorder projection $[p_1] \leq_p \dots \leq_p [p_m]$ of a block with the sequence $\text{parikh}([p_1]), \dots, \text{parikh}([p_m])$ over $\text{parikh}_k(\Sigma)$. We observe that the image of a block B in the linear order projection and in the preorder projection of S is a contiguous interval. See Figure 3 for an example of blocks.

A *partial run* of an automaton \mathcal{A} is a pair (p, q) of states of \mathcal{A} . Two partial runs $r = (p, q)$ and $s = (q, r)$ can be *concatenated* (or *connected*) to a partial run $t = r \cdot s = (p, r)$. For a partial run $r = (p, q)$, p and q are called *start* and *end* of the partial run, respectively.

The proof of the following lemma is rather technical and can be found in the Appendix.

► **Lemma 9.** *For a given finite state transducer \mathcal{B} and two states s, t of \mathcal{B} , there is a finite state automaton accepting exactly those sequences p over $\text{parikh}_k(\Pi)$ that satisfy the following conditions:*

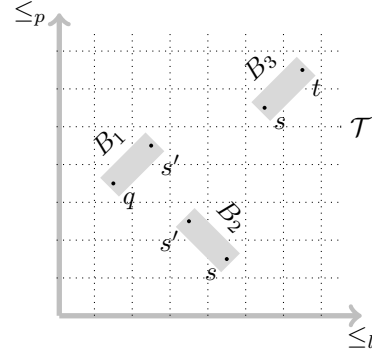
- *There is exactly one position of p that contains an element marked with $L \in \{L^+, L^-\}$. Analogously for $R \in \{R^+, R^-\}$.*
- *There is a block B and a run r of \mathcal{B} on B such that*
 - *s is the start state of r and t is the final state of r .*
 - *the preorder projection of the output of \mathcal{B} with run r is p .*

Linear order case. We introduce some of the ideas for the k -ODA case by warming up with the 1-ODA case. In 1-bounded $(+1_l; +1_p, \leq_p)$ -structures the preorder is a linear order, hence elements from $\text{parikh}_1(\Pi)$ can be identified with Π . Further, in the 1-bounded case $L(B)$ and $R(B)$ are the highest (or lowest) and lowest (or highest) elements of B with respect to \leq_p .

► **Proposition 4.** *The emptiness problem of 1-ODA can be reduced to the emptiness problem of multicounter automata.*

Proof sketch. Given a 1-ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$, we construct a multicounter automaton \mathcal{M} such that \mathcal{A} accepts a 1-bounded $(+1_l; +1_p, \leq_p)$ -structure if and only if \mathcal{M} accepts a sequence over $\text{parikh}_1(\Pi)$.

For a $(+1_l; +1_p, \leq_p)$ -structure \mathcal{S} with blocks B_1, \dots, B_l (ordered by the linear order), the automaton \mathcal{M} on input $\text{pp}(\mathcal{S})$ works as follows. While running over $\text{pp}(\mathcal{S})$ it guesses and verifies a run of \mathcal{C} .



■ **Figure 4** How a 1-ODA is simulated by a multicounter automaton \mathcal{M} . When \mathcal{M} reaches the solid line \mathcal{T} , the counter for (q, s) is one. When starting to read block B_3 , the counter for (q, s) is decremented and (q, t) is stored in the state.

In parallel \mathcal{M} constructs a run of \mathcal{B} . Since \mathcal{M} runs on $\text{pp}(\mathcal{S})$, it sees a permutation $B_{\pi(1)}, \dots, B_{\pi(l)}$ of the blocks B_1, \dots, B_l . Intuitively, for the construction of a run of \mathcal{B} , this permutation has to be properly rearranged. In the rearrangement the end of one block and the beginning of the next block are not allowed to be \leq_p -close. That is for every block B_i with R^+ -marked rightmost element v , the leftmost element u of the following block in the rearrangement has to be far above v with respect to \leq_p . In particular u has to be marked with L^- .

The multicounter automaton rearranges the blocks as follows. It guesses those positions in $\text{pp}(\mathcal{S})$, in which the blocks B_1, \dots, B_l start and end. The construction of a run of \mathcal{B} comprises, for every block B_i , the following two simultaneous steps:

- \mathcal{M} guesses a partial run $r = (s, t)$ such that \mathcal{B} can reach B_i in state s and leave B_i in state t . This can be verified using Lemma 9.
- \mathcal{M} tries to concatenate r with partial runs already seen in accordance to L^+, L^-, R^+ and R^- labels. The resulting partial run is saved.

For the second step, it is necessary to remember how often every partial run has been seen so far. Therefore \mathcal{M} uses counters from $Q \times Q$.

We make this more precise. Intuitively, the start and end of the partial run $r = (s, t)$ guessed for B_i will be concatenated, if the blocks B_{i-1} and B_{i+1} have already occurred in the preorder projection. We assume that B_i is neither the first nor the last block, i.e. $i \neq 1$ and $i \neq l$. Further, we assume that the leftmost position u of B_i occurs before the rightmost position v of B_i in the preorder projection. When encountering u , the multicounter automaton \mathcal{M} stores a partial run r' in its state, depending on whether u is labeled with L^+ or L^- :

- If u is marked by L^+ , the block B_{i-1} did not occur in the preorder projection yet. The partial run $r' = r$ is stored in the state.
- If u is marked by L^- , the block B_{i-1} did already occur in the preorder projection. Therefore \mathcal{M} can guess a partial run (s', s) that ends at the leftmost position of B_i and is already saved in the counters (see Figure 4). If the last position in the preorder was an R^+ position with partial run (s', s) , then \mathcal{M} makes sure that the counter corresponding to (s', s) is at least 2 (to ensure that a run (s', s) can be chosen such that the block corresponding to that run is far away with respect to the preorder, and therefore satisfying the L^- and R^+ labeling). Now, \mathcal{M} stores the partial run $r' = (s', s) \cdot r$ in its state and decrements the counter (s', s) .

\mathcal{M} behaves similarly when encountering v , see the Appendix for more details.

The cases where B_i is the first or last block as well as the case when \mathcal{M} encounters the rightmost position v of B_i first, can be settled similarly.

\mathcal{M} accepts if, after reading the whole input sequence, only one counter (i, f) is equal to 1 where i is the initial state and f is some final state of \mathcal{B} . ◀

Bounded preorder case. For k -bounded preorders with $k > 1$ we extend the construction of Proposition 4. However, the situation in the k -bounded case is more complicated for several reasons:

- While reading the preorder projection, the multicounter automaton has to process several blocks at once.
- The L and R markers can appear anywhere in the preorder projection of a block.
- The interaction of L and R markers of several blocks has to be considered.

Those problems can be solved.

► **Theorem 10.** *The emptiness problem of k -ODA can be reduced to the emptiness problem of multicounter automata.*

Proof sketch. Again, for a given k -ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ we construct a multicounter automaton \mathcal{M} such that \mathcal{A} accepts a k -bounded $(+1_l; +1_p, \leq_p)$ -structure if and only if \mathcal{M} accepts a sequence over $\text{parikh}_k(\Sigma)$.

Blocks whose start has been read, but whose end still needs to be read, are called *active*. Up to k blocks can now overlap in the preorder projection (see e.g. row 4 in Figure 3). Thus there are at most k active blocks. The multicounter automaton \mathcal{M} has a sub-automaton for every active block.

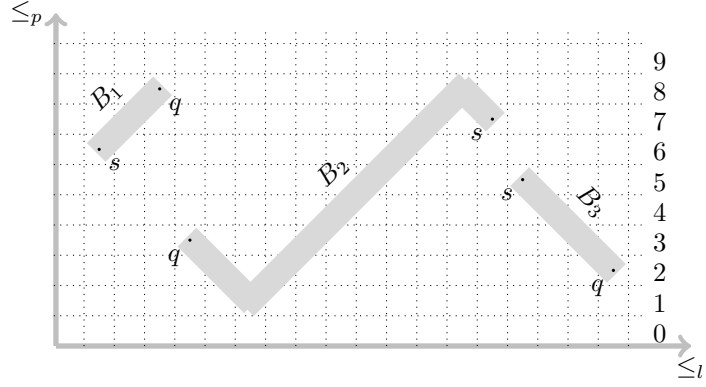
Consider an input $(+1_l; +1_p, \leq_p)$ -structure \mathcal{S} with blocks B_1, \dots, B_l (ordered by the linear order). The automaton \mathcal{M} guesses and verifies a run of \mathcal{C} on $\text{pp}(\mathcal{S})$. In parallel \mathcal{M} constructs a run of \mathcal{B} . For every input symbol $p \in \text{parikh}_k(\Sigma)$ this comprises the following steps:

- Guess which blocks start at the current position. For every newly started active block B_i , \mathcal{M} guesses a partial run $r = (s, t)$ such that \mathcal{B} can reach B_i in state s and leave B_i in state t . \mathcal{M} creates a new sub-automaton, namely the finite state automaton from Lemma 9 with states s and t .
- Guess which active blocks end at the current position. Reject if there is a block that ends and whose corresponding sub-automaton is not in a final state.
- Guess a partition of p into p_1, \dots, p_m such that p_i belongs to the active block A_i . Simulate one step of the sub-automata \mathcal{A}_i corresponding to A_i with p_i .
- When p_i contains L^{+-} , L^{-} , R^{+-} or R^{-} -labeled positions the partial run assigned to B_i is adjusted properly.

Again, \mathcal{M} uses counters from $Q \times Q$ for performing the third step.

As before \mathcal{M} uses the counter (s, t) for saving the number of partial runs of \mathcal{B} from s to t that have already been seen during the simulation. However, some partial runs will be *cached* in the states of \mathcal{M} , namely runs where one, either start or end state corresponds to an active block. Therefore \mathcal{M} maintains for every cached run r , two pointers $L(r)$ and $R(r)$ that contain the active block that corresponds to the start or end state of r , respectively, or $'-'$ if there is no corresponding active block. (Since there are at most k active blocks, such pointers can be stored.)

The start of a cached run r is *connectable*, if $L(r) = '-'$ and the corresponding position occurred on position i of the input and currently position $j > i+1$ is processed. Similarly, the start point of a run corresponding to counter r is connectable, if counter r has a value greater than the number of increments on the last position due to L -labeled elements. Analogously



■ **Figure 5** How a k -ODA is simulated by a multicounter automaton \mathcal{M} . See Example 11 for an explanation.

for ends of runs. The multicounter automaton \mathcal{M} can easily maintain information about connectable end points of runs. 1

The construction of a complete run of \mathcal{B} is similar to the 1-bounded case but more involved, see the Appendix for a detailed description. 2

► **Example 11.** We look at an example of how the multicounter automaton from Theorem 10 constructs a run of \mathcal{B} when reading $\text{pp}(\mathcal{S})$ where \mathcal{S} is as given in Figure 5. For simplicity we assume that to the left and right of B_1 and B_3 there are more blocks that will appear much later in the preorder projection. 3

The run of \mathcal{B} is constructed as follows: 4

- (Row 1) The partial run (q, s) is guessed for block B_2 and saved in the cache. 5
- (Row 2) The partial run (s, q) is guessed for block B_3 and saved in the cache. 6
- (Row 3) The partial run (s, q) is removed from the cache, the counter (s, q) is incremented. 7
- (Row 4) The partial run (s, q) is guessed for block B_1 and saved in the cache. 8
- (Row 5) The R^- marking of block B_2 is encountered. There are two choices, either to connect the corresponding run to the cached run of block B_1 , or to the stored run of block B_3 . However, the cached run of block B_1 is marked unconnectable (as it occurred in the last row). Thus counter (s, q) is decremented and counter $r = (q, s) \cdot (s, q)$ is incremented. 9
- (Row 6) The R^- marking of block B_1 is encountered. Thus the multicounter automaton connects the run (s, q) to a run (q, q) . Therefore it decreases counter (q, q) , increases counter (s, q) and removes (s, q) from the cache. 10

Lower bounds. Although decidable, the complexity for satisfiability of $\text{FO}^2(\leq_{l_1}, +1_{l_1}; +1_{l_2})$ is very high. 11

► **Proposition 5.** Satisfiability of $\text{FO}^2(\leq_{l_1}, +1_{l_1}; +1_{l_2})$ is at least as hard as the reachability problem for vector addition systems. 12

Since the logic $\text{FO}^2(+1_l; +1_p, \leq_p)$ allows for axiomatizing 1-boundedness, we have the following corollary. 13

► **Corollary 12.** Satisfiability of $\text{FO}^2(+1_l; +1_p, \leq_p)$ over k -bounded ordered data words is at least as hard as the reachability problem for vector addition systems. 14

4 Undecidability Results for Two-Dimensional Ordered Structures

This section aims at filling the remaining gaps for two-dimensional ordered structures. We refer the reader to Figure 6 for a summary of results obtained in the literature and in this article. Proofs that are not presented here can be found in the Appendix.

It is not surprising that FO^2 with two additional preorder successor relations is undecidable, as those allow for encoding a grid. The proof is straight forward.

► **Proposition 6.** *Finite satisfiability of two-variable logic with two additional preorder successor relations, that is $\text{FO}^2(+1_{p_1}; +1_{p_2})$, is undecidable.*

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 fro 2-bounded structures. This gives a tight bound on the extensibility of the decidability of $\text{FO}^2(+1_l; +1_p, \leq_p)$ on k -bounded structures to structures with two preorders.

► **Proposition 7.** *Finite satisfiability of two-variable logic with two additional 2-bounded preorder successor relations is undecidable.*

We denote the relation $+1_l^2$ bei $+2_l$. The following corollary slightly improves Theorem 4 from [12]. The proof uses that a 2-bounded preorder successor relation can be projectively characterized using relations $+1_l$ and $+2_l$.

► **Corollary 13.** *Finite satisfiability of $\text{FO}^2(+1_{l_1}, +2_{l_1}; +1_{l_2}, +2_{l_2})$ is undecidable.*

The following propositions complement results from [2] and [16]. The proofs use similar methods as used in those works.

► **Proposition 8.** *Finite satisfiability of two-variable logic with a linear order, its corresponding successor relation and an additional preorder successor relation, i.e. $\text{FO}^2(+1_l, \leq_l; +1_p)$, is undecidable.*

► **Proposition 9.** *Finite satisfiability of two-variable logic with one additional preorder successor relation and one additional preorder relation, i.e. $\text{FO}^2(+1_{p_1}; \leq_{p_2})$, is undecidable.*

5 Discussion

The current status of research on two-variable logic with additional successor and order relations is summarized in Figure 6.

We have seen that two-variable logic with an additional linear order successor and a k -bounded preorder relation with its successor relation remains decidable. Whether this can be generalized remains open.

► **Open Question 1.** *Is finite satisfiability of $\text{FO}^2(+1_{l_1}; +1_{p_2}, \leq_{p_2})$ decidable?*

Further, it is open whether there is some m such that $\text{FO}^2(+1_{l_1}; \dots; +1_{l_m})$ is undecidable. On first view, one might think that there should be such a m . However, a method for proving undecidability of $\text{FO}^2(+1_{l_1}; \dots; +1_{l_m})$ should not extend to $\text{FO}^2(F_1, \dots, F_m)$ 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 $\text{FO}^2(F_1, \dots, F_m)$ is decidable since one can express that some arbitrary interpreted predicate R is a permutation by using two-variable logic with counting quantifiers which in turn is decidable by [7]. This is an observation by Juha Kontinen.

Logic	Complexity (lower/upper)	Comments
One linear order		
$\text{FO}^2(+1_l)$	NEXPTIME-complete	[5]
$\text{FO}^2(\leq_l)$	NEXPTIME-complete	[5]
$\text{FO}^2(+1_l, \leq_l)$	NEXPTIME-complete	[5]
One total preorder		
$\text{FO}^2(+1_p)$	EXPSpace-complete	EXPCORRIDORTILING
$\text{FO}^2(\leq_p)$	NEXPTIME/EXPSpace	
$\text{FO}^2(+1_p, \leq_p)$	EXPSpace-complete	[16]
Two linear orders		
$\text{FO}^2(+1_{l_1}; +1_{l_2})$	NEXPTIME/2-NEXPTIME	[12]
$\text{FO}^2(+1_{l_1}; \leq_{l_2})$	NEXPTIME/EXPSpace	[16]
$\text{FO}^2(+1_{l_1}, \leq_{l_1}; +1_{l_2})$	VASS-REACHABILITY/Decidable	★, Cor. 2 and Prop. 5
$\text{FO}^2(+1_{l_1}, \leq_{l_1}; \leq_{l_2})$	NEXPTIME/EXPSpace	[16]
$\text{FO}^2(+1_{l_1}, \leq_{l_1}; +1_{l_2}, \leq_{l_2})$	Undecidable	[12]
Two total preorders		
$\text{FO}^2(+1_{p_1}, +1_{p_2})$	Undecidable	★, Prop. 6 and 7
$\text{FO}^2(+1_{p_1}; \leq_{p_2})$	Undecidable	★, Prop. 9
$\text{FO}^2(\leq_{p_1}; \leq_{p_2})$	Undecidable	[15]
One linear order and one total preorder		
$\text{FO}^2(+1_{l_1}; +1_{p_2})$?	★ Special case, Theorem 1
$\text{FO}^2(+1_{l_1}, \leq_{l_1}; +1_{p_2})$	Undecidable	★, Prop. 8
$\text{FO}^2(+1_{l_1}, \leq_{l_1}; \leq_{p_2})$	Undecidable	[2]
$\text{FO}^2(+1_{l_1}; +1_{p_2}, \leq_{p_2})$?	★, Special case, Theorem 1
$\text{FO}^2(\leq_{l_1}; +1_{p_2}, \leq_{p_2})$	EXPSpace-complete	[16]
Many orders		
$\text{FO}^2(\leq_{l_1}; \leq_{l_2}; \leq_{p_3})$	Undecidable	[15]
$\text{FO}^2(\leq_{l_1}; \dots; \leq_{l_s})$	Undecidable	[14]
$\text{FO}^2(+1_{l_1}; +1_{l_2}; +1_{l_3}; \dots)$?	

■ **Figure 6** 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. ★-denotes results in this paper.

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

Temporal logics on data words have seen much research recently [3, 4, 9]. 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 [18] 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 3.** *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 usual order on 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 [16]. The approach taken here for $\text{FO}^2(+1_l; +1_p, \leq_p)$ can most likely be generalized to the incomplete successor setting.

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 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 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.
- 4 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.
- 5 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.
- 6 Erich Grädel and Martin Otto. On logics with two variables. *Theor. Comput. Sci.*, 224(1-2):73–113, 1999.
- 7 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *LICS*, pages 306–317, 1997.
- 8 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.
- 9 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 fuer Informatik, 2010.
- 10 Emanuel Kieronski and Martin Otto. Small substructures and decidability issues for first-order logic with two variables. In *LICS*, pages 448–457, 2005.
- 11 Emanuel Kieronski and Lidia Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *LICS*, pages 123–132, 2009.
- 12 Amaldev Manuel. Two orders and two variables. In *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 513–524, 2010.
- 13 M. Mortimer. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 21:135–140, 1975.
- 14 Martin Otto. Two variable first-order logic over ordered domains. *J. Symb. Log.*, 66(2):685–702, 2001.
- 15 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.
- 16 Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. Submitted to Logical Methods in Computer Science, 2011.
- 17 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, pages 41–57, 2006.
- 18 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 fuer Informatik, 2011.

Appendix

For the convenience of the reader we repeat proofs completely.

Proofs from Section 3.1

Lemma 6. *There is a k -ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ such that \mathcal{A} accepts a k -bounded structure \mathcal{S} if and only if \mathcal{B} outputs $\text{pro}(\mathcal{S})$, i.e. there is a k -ODA that can guess and verify the profile of a structure.*

Proof. Let $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ be a k -ODA such that \mathcal{B} guesses profiles for every element and also forwards the preorder marking to the automaton \mathcal{C} .

We show how \mathcal{B} and \mathcal{C} can, for every type τ and all elements u , verify that

- if u is labeled with τ , then there is a v such that (u, v) is of type τ .
- if u is not labeled with τ , then there is no v such that (u, v) is of type τ .

We fix a type $\tau(x, y)$ and assume that τ requires the unary types of x and y to be σ and σ' , respectively. Basically, by labelling u with $\tau(x, y)$, \mathcal{B} claims that there is an element v labeled with σ' in a certain direction from u (which is specified by the binary literals in $\tau(x, y)$). Let $\tau_l \subseteq \tau(x, y)$ (resp. $\tau_p(x, y) \subseteq \tau(x, y)$) be maximal with $\tau_l \subseteq \Gamma_l$ (resp. $\tau_p \subseteq \Gamma_p$).

Now, we show how the following main cases can be verified:

- i) $|\tau_l(x, y)| \leq 1$
- ii) $|\tau_p(x, y)| \leq 1$
- iii) $\tau_l = \infty$ and $|\tau_p(x, y)| = \infty$

See Figure 2 for a graphical illustration of those cases. The cases i) and ii) can be found in the Appendix.

For iii) assume $\tau_l = \infty$ and $\tau_p(x, y) = +\infty$. Firstly, we observe the following. There is some bound $m \in \mathbb{N}$ such that any σ -position u with at least m later occurrences of σ' (with respect to the preorder projection) has profile τ . This follows from the fact that there only boundedly many elements that are $+1_l$ - or $+1_p$ -close to u . Now, \mathcal{B} guesses the last m (with respect to \leq_p) σ' -positions v_1, \dots, v_m and colors them with new labels $\sigma'_1, \dots, \sigma'_m$ (if there are less σ' -positions, \mathcal{B} colors them distinctly). Further, \mathcal{B} colors every position u with tuples $(\tau_1, \sigma_1), \dots, (\tau_m, \sigma_m)$ such that $\tau_i = \tau(u, v_i)$. \mathcal{B} and \mathcal{C} can verify this coloring since $\{v_1, \dots, v_m\}$ is finite (see Example 3).

Now, checking that \mathcal{B} guessed τ correctly for all positions is easy. Let u be an arbitrary position. If u is labeled with τ , then \mathcal{B} checks that u is labeled with some τ_i , $i \in \{1, \dots, m\}$ with $\tau = \tau_i$ (then v_i is a witness with $\tau(u, v_i)$). If u is not labeled with τ then \mathcal{C} verifies that there are at most m σ' -positions after u (with respect to \leq_p) and that $\tau_i \neq \tau$ for all τ_i . Similarly for the other cases of iii). This completes the proof. ◀

Proposition 2. *Given a k -ODA \mathcal{A} there is a sentence $\varphi_{\mathcal{A}} \in \text{EMSO}^2(+1_l; +1_p, \leq_p)$ such that $L(\mathcal{A}) = L_k(\varphi_{\mathcal{A}})$.*

Proof. Let $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ be a k -ODA. Besides using predicates from Σ , the formula $\varphi_{\mathcal{A}}$ uses, among others, second order variables from Π , Γ_p , $\text{parikh}_k(\Pi)$ as well as $Y = \{Y_1, \dots, Y_k\}$. The variables from Γ_p are used to guess the preorder marking for every element. Checking that this marking is correct can be easily achieved by using the binary predicates $+1_l$, $+1_p$ and \leq_p .

The variables from Π are used to guess the output from \mathcal{B} . Since \mathcal{B} is a finite state transducer there is a EMSO²-formula $\varphi_{\mathcal{B}}$ that encodes a successful run of \mathcal{B} .

The preorder automaton \mathcal{C} runs on k -bounded parikh vectors of Π . For every position the parikh vector of its \sim_p -class can be guessed, and consistency with the Π -labelling verified by using variables from Y . Then, a run of \mathcal{C} can be encoded in an EMSO²-sentence $\varphi_{\mathcal{C}}$.

The formula $\varphi_{\mathcal{A}}$ is the conjunction of all these formulas. \blacktriangleleft

Lemma 7. *For every sentence φ of the form $\forall x \forall y \psi$ with ψ quantifier-free there is a k -ODA accepting exactly the k -bounded models of φ .*

Proof. It is straight forward to convert φ into a formula of the form

$$\bigwedge_{\sigma, \tau \in \mathcal{U}} \forall x \forall y (\sigma(x) \wedge \tau(y) \rightarrow \gamma_{\sigma, \tau}(x, y))$$

where $\gamma_{\sigma, \tau}$ are disjunctions of binary types and \mathcal{U} denotes the set of unary types. Note that for a quantifier-free formula $\varphi(x, y)$ in disjunctive normal form, an equivalent quantifier-free formula in DNF can be constructed such that, in every disjunct, the unary types of x and y are fully specified. Further it can be assured that all combinations of unary types for x and y occur in some disjunct. Such a DNF is said to be *complete*.

The formula ψ can be transformed into a complete DNF. By sorting the disjuncts by the unary types of x and y we obtain a formula

$$\chi = \forall x \forall y \bigvee_{\sigma, \tau \in \mathcal{U}} \sigma(x) \wedge \tau(y) \wedge \gamma_{\sigma, \tau}(x, y)$$

where $\gamma_{\sigma, \tau}$ are disjunctions of binary types and \mathcal{U} denotes the set of unary types. This formula is equivalent to the formula

$$\bigwedge_{\sigma, \tau \in \mathcal{U}} \forall x \forall y (\sigma(x) \wedge \tau(y) \rightarrow \gamma_{\sigma, \tau}(x, y))$$

Again, every conjunct can be checked separately due to closure under intersection. A single condition $\forall x \forall y (\sigma(x) \wedge \tau(y) \rightarrow \gamma(x, y))$ allows for elements a and b with labels σ and τ to be related in ways that are allowed by γ . That is, certain binary types – namely those that do not occur in the disjunction – are forbidden for a and b . This can be checked by a k -ODA using the information contained in the profiles. For example, if $+1_l(x, y) \wedge x \leq_p y \wedge \neg +1_p(x, y)$ is forbidden by γ , then a k -ODA has to check that, for every element labeled with σ , the type $+1_l(x, y) \wedge x \leq_p y \wedge \neg +1_p(x, y) \wedge \sigma(x) \wedge \tau(y)$ is not contained in the profile. \blacktriangleleft

Lemma 8. *For every sentence φ of the form $\forall x \exists y \chi$ with χ quantifier-free there is a k -ODA accepting exactly the k -bounded models of φ .*

Proof. As in the previous lemma, by using complete DNF and sorting by the complete unary types of x , ψ_i can be transformed into a formula

$$\forall x \exists y \bigvee_{\sigma \in \mathcal{U}} (\sigma(x) \wedge \bigvee_{\tau \in \mathcal{U}} (\tau(y) \wedge \gamma_{\sigma, \tau}(x, y)))$$

which in turn can be easily translated into

$$\bigwedge_{\sigma \in \mathcal{U}} \forall x (\sigma(x) \rightarrow \exists y \bigvee_{\tau \in \mathcal{U}} (\tau(y) \wedge \gamma_{\sigma, \tau}(x, y)))$$

A single condition $\forall x(\sigma(x) \rightarrow \exists y \bigvee_{\tau \in \mathcal{U}}(\tau(y) \wedge \gamma_{\sigma, \tau}(x, y)))$ can be checked by a k -ODA $(\mathcal{B}, \mathcal{C})$ as follows. For every occurrence of σ , the automaton \mathcal{B} guesses which $\tau \in \Sigma$ witnesses the correctness of the formula and in which direction it can be found. The correctness of this guess can be verified using profile information. ◀

Proofs from Section 3.2

Lemma 9. *For a given finite state transducer \mathcal{B} and two states s, t of \mathcal{B} , there is a finite state automaton accepting exactly those sequences p over $\text{parikh}_k(\Pi)$ that satisfy the following conditions:*

- *There is exactly one position of p that contains an element marked with $L \in \{L^+, L^-\}$. Analogously for $R \in \{R^+, R^-\}$.*
- *There is a block B and a run r of \mathcal{B} on B such that*
 - *s is the start state of r and t is the final state of r .*
 - *the preorder projection of the output of \mathcal{B} with run r is p .*

Proof sketch. For a finite state transducer \mathcal{B} , we describe how to construct the desired finite state automaton \mathcal{A} . Therefore we describe how \mathcal{A} works on an input sequence $p = p_1, \dots, p_m$ over $\text{parikh}_k(\Pi)$. Note that every p_i can be thought of as one preorder equivalence class where p_i saves, for every $\sigma \in \Sigma$, the number of σ -labeled elements in that equivalence class. We denote the equivalence class corresponding to p_i by $[p_i]$.

The following observation is crucial. Let B be a bloc with preorder projection $[p_1], \dots, [p_m]$. The restriction of \mathcal{B} to elements from $[p_1], \dots, [p_l]$, for some l , yields at most $k + 1$ blocks B_1, \dots, B_{k+1} . (Assume there are more than $k + 1$ blocks, then one $[p_j]$ with $j > l$ has to contain more than k elements.) The automaton will maintain, for every B_i , a partial run (p_i, q_i) such that \mathcal{B} arrives B_i in state p_i and leaves B_i in q_i .

Thus \mathcal{A} stores a multiset of at most $k + 1$ partial runs in its memory. At the beginning no partial runs are stored. Now \mathcal{A} performs, for every input symbol p_i , one round of the following three steps.

First \mathcal{A} guesses, for every element u in $[p_i]$, a partial run (p, q) and a symbol $\sigma \in \Sigma$ such that when \mathcal{B} is in state p and reads σ , it outputs the label of u and goes into state q . If u is marked with $L \in \{L^+, L^-\}$ (or $R \in \{R^+, R^-\}$), a partial run (s, q) (or (q, t)) is guessed, where q is an arbitrary state from \mathcal{B} . In this case the start (end) of this partial run is marked as *dead* and will never be connected in what follows.

Secondly, \mathcal{A} guesses how to connect partial runs from the first step. Those are runs of \mathcal{B} on sequences of positions that are equivalent with respect to \sim_p .

Thirdly, to starts and ends (that are not marked dead) of partial runs stored in the memory, \mathcal{A} connects a partial run obtained in the second step. Note that one partial run obtained in the second step can be connected to none, one or two partial runs from the memory. \mathcal{A} stores the resulting runs in its state.

After every round \mathcal{A} makes sure that at most k partial runs are stored. \mathcal{A} accepts, when only the partial run (s, t) is stored after the last round.

The proof of correctness will appear in the full paper. ◀

Proposition 4. *The emptiness problem of 1-ODA can be reduced to the emptiness problem of multicounter automata.*

Proof sketch. Given a 1-ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$, we construct a multicounter automaton \mathcal{M} such that \mathcal{A} accepts a 1-bounded $(+1_l; +1_p, \leq_p)$ -structure if and only if \mathcal{M} accepts a sequence over $\text{parikh}_1(\Pi)$.

For a $(+1_l; +1_p, \leq_p)$ -structure \mathcal{S} with blocks B_1, \dots, B_l (ordered by the linear order), the automaton \mathcal{M} on input $\text{pp}(\mathcal{S})$ works as follows. While running over $\text{pp}(\mathcal{S})$ it guesses and verifies a run of \mathcal{C} .

In parallel \mathcal{M} constructs a run of \mathcal{B} . Since \mathcal{M} runs on $\text{pp}(\mathcal{S})$, it sees a permutation $B_{\pi(1)}, \dots, B_{\pi(l)}$ of the blocks B_1, \dots, B_l . Intuitively, for the construction of a run of \mathcal{B} , this permutation has to be properly rearranged. In the rearrangement the end of one block and the beginning of the next block are not allowed to be \leq_p -close. That is for every block B_i with R^+ -marked rightmost element v , the leftmost element u of the following block in the rearrangement has to be far above v with respect to \leq_p . In particular u has to be marked with L^- .

The multicounter automaton rearranges the blocks as follows. It guesses those positions in $\text{pp}(\mathcal{S})$, in which the blocks B_1, \dots, B_l start and end. The construction of a run of \mathcal{B} comprises, for every block B_i , the following two simultaneous steps:

- \mathcal{M} guesses a partial run $r = (s, t)$ such that \mathcal{B} can reach B_i in state s and leave B_i in state t . This can be verified using Lemma 9.
- \mathcal{M} tries to concatenate r with partial runs already seen in accordance to L^+, L^-, R^+ and R^- labels. The resulting partial run is saved.

For the second step, it is necessary to remember how often every partial run has been seen so far. Therefore \mathcal{M} uses counters from $Q \times Q$.

We make this more precise. Intuitively, the start and end of the partial run $r = (s, t)$ guessed for B_i will be concatenated, if the blocks B_{i-1} and B_{i+1} have already occurred in the preorder projection. We assume that B_i is neither the first nor the last block, i.e. $i \neq 1$ and $i \neq l$. Further, we assume that the leftmost position u of B_i occurs before the rightmost position v of B_i in the preorder projection. When encountering u , the multicounter automaton \mathcal{M} stores a partial run r' in its state, depending on whether u is labeled with L^+ or L^- :

- If u is marked by L^+ , the block B_{i-1} did not occur in the preorder projection yet. The partial run $r' = r$ is stored in the state.
- If u is marked by L^- , the block B_{i-1} did already occur in the preorder projection. Therefore \mathcal{M} can guess a partial run (s', s) that ends at the leftmost position of B_i and is already saved in the counters (see Figure 4). If the last position in the preorder was an R^+ position with partial run (s', s) , then \mathcal{M} makes sure that the counter corresponding to (s', s) is at least 2 (to ensure that a run (s', s) can be chosen such that the block corresponding to that run is far away with respect to the preorder, and therefore satisfying the L^- and R^+ labeling). Now, \mathcal{M} stores the partial run $r' = (s', s) \cdot r$ in its state and decrements the counter (s', s) .

When encountering v , block B_i is completely read and a partial run r'' , that depends on whether v is labeled with R^- or R^+ , is saved in the counter r'' . The partial run r'' is obtained as follows:

- If v is marked by R^+ , the block B_{i+1} did not occur in the preorder projection yet. The counter for the partial run $r'' = r'$ is incremented.
- If v is marked by R^- , the block B_{i+1} did already occur in the preorder projection. Therefore \mathcal{M} can guess a partial run (t, t') that starts at the rightmost position of B_i and is already saved in the counters. If the last position in the preorder was an L^- position with partial run (s', s) , then \mathcal{M} makes sure that the counter corresponding to

(t, t') is at least 2. The counter for the partial run $r'' = r' \cdot (t, t')$ is incremented and the counter for (t, t') is decremented.

The cases where B_i is the first or last block as well as the case when \mathcal{M} encounters the rightmost position v of B_i first, can be settled similarly.

\mathcal{M} accepts if, after reading the whole input sequence, only one counter (i, f) is equal to 1 where i is the initial state and f is some final state of \mathcal{B} . ◀

Theorem 10. *The emptiness problem of k -ODA can be reduced to the emptiness problem of multicounter automata.*

Proof sketch. Again, for a given k -ODA $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ we construct a multicounter automaton \mathcal{M} such that \mathcal{A} accepts a k -bounded $(+1_l; +1_p, \leq_p)$ -structure if and only if \mathcal{M} accepts a sequence over $\text{parikh}_k(\Sigma)$.

Blocks whose start has been read, but whose end still needs to be read, are called *active*. Up to k blocks can now overlap in the preorder projection (see e.g. row 4 in Figure 3). Thus there are at most k active blocks. The multicounter automaton \mathcal{M} has a sub-automaton for every active block.

Consider an input $(+1_l; +1_p, \leq_p)$ -structure \mathcal{S} with blocks B_1, \dots, B_l (ordered by the linear order). The automaton \mathcal{M} guesses and verifies a run of \mathcal{C} on $\text{pp}(\mathcal{S})$. In parallel \mathcal{M} constructs a run of \mathcal{B} . For every input symbol $p \in \text{parikh}_k(\Sigma)$ this comprises the following steps:

- Guess which blocks start at the current position. For every newly started active block B_i , \mathcal{M} guesses a partial run $r = (s, t)$ such that \mathcal{B} can reach B_i in state s and leave B_i in state t . \mathcal{M} creates a new sub-automaton, namely the finite state automaton from Lemma 9 with states s and t .
- Guess which active blocks end at the current position. Reject if there is a block that ends and whose corresponding sub-automaton is not in a final state.
- Guess a partition of p into p_1, \dots, p_m such that p_i belongs to the active block A_i . Simulate one step of the sub-automata \mathcal{A}_i corresponding to A_i with p_i .
- When p_i contains L^+ -, L^- -, R^+ - or R^- -labeled positions the partial run assigned to B_i is adjusted properly.

Again, \mathcal{M} uses counters from $Q \times Q$ for performing the third step.

As before \mathcal{M} uses the counter (s, t) for saving the number of partial runs of \mathcal{B} from s to t that have already been seen during the simulation. However, some partial runs will be *cached* in the states of \mathcal{M} , namely runs where one, either start or end state corresponds to an active block. Therefore \mathcal{M} maintains for every cached run r , two pointers $L(r)$ and $R(r)$ that contain the active block that corresponds to the start or end state of r , respectively, or $'-'$ if there is no corresponding active block. (Since there are at most k active blocks, such pointers can be stored.)

The start of a cached run r is *connectable*, if $L(r) = '-'$ and the corresponding position occurred on position i of the input and currently position $j > i+1$ is processed. Similarly, the start point of a run corresponding to counter r is connectable, if counter r has a value greater than the number of increments on the last position due to L -labeled elements. Analogously for ends of runs. The multicounter automaton \mathcal{M} can easily maintain information about connectable end points of runs.

We describe how a complete run of \mathcal{B} is successively constructed, by outlining what happens if \mathcal{M} processes block B_i . Intuitively, the partial run $r = (s, t)$ guessed for B_i will be connected to those partial runs that have already been seen. We assume that B_i is

neither the first nor the last block, i.e. $i \neq 1$ and $i \neq l$. Further, we assume that the leftmost position u of B_i occurs before the rightmost position v of B_i in the preorder projection.

When encountering u , the multicounter automaton \mathcal{M} proceeds as follows

- If u is marked by L^+ , the R^- -labeled position of block B_{i-1} did not occur in the preorder projection yet. The partial run $r' = (s, t)$ is cached in the state with $L(r') = R(r') = B_i$.
- If u is marked by L^- , the R^+ -labeled position of block B_{i-1} did already occur in the preorder projection. Now, \mathcal{M} guesses whether the run q corresponding to B_{i-1} is cached in the state or saved in the counter q :
 - If q is cached and $R(q)$ is connectable, then q is replaced by the partial run $r' = q \cdot r$ with $L(r') = L(q)$ and $R(r') = B_i$.
 - If q is saved in counter q and counter q is connectable, then counter q is decremented and the partial run $r' = q \cdot r$ with $L(r') = -'$ and $R(r') = B_i$ is cached.

When v is encountered the multicounter automaton \mathcal{M} proceeds as follows. Let r' be the run with $R(r') = B_i$. (Note that $L(r')$ is not necessarily the same as the one saved in the last step.)

- If v is marked by R^+ , the L^- -labeled position of block B_{i+1} did not occur in the preorder projection yet. \mathcal{M} distinguishes whether $L(r') = -'$ or $L(r') = B_j$, for some j .
 - If $L(r') = -'$ then the counter r' is incremented.
 - If $L(r') = B_j$ then r' is cached again with $L(r') = B_j$ and $L(r') = -'$.
- The case if v is marked by R^- is analogous.

When several blocks reach their L -, R -position simultaneously, the procedure above is done to all those blocks. The corresponding starts and ends of blocks cannot be connected.

The cases where B_i is the first or last blocks as well as the case when \mathcal{M} encounters the rightmost position v of B_i first can be settled similarly.

See Example 11 for a concrete construction. ◀

Proposition 5. *Satisfiability of $\text{FO}^2(\leq_{l_1}, +1_{l_1}; +1_{l_2})$ is at least as hard as the reachability problem for vector addition systems.*

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 $+1_{l_2}$, the result follows.

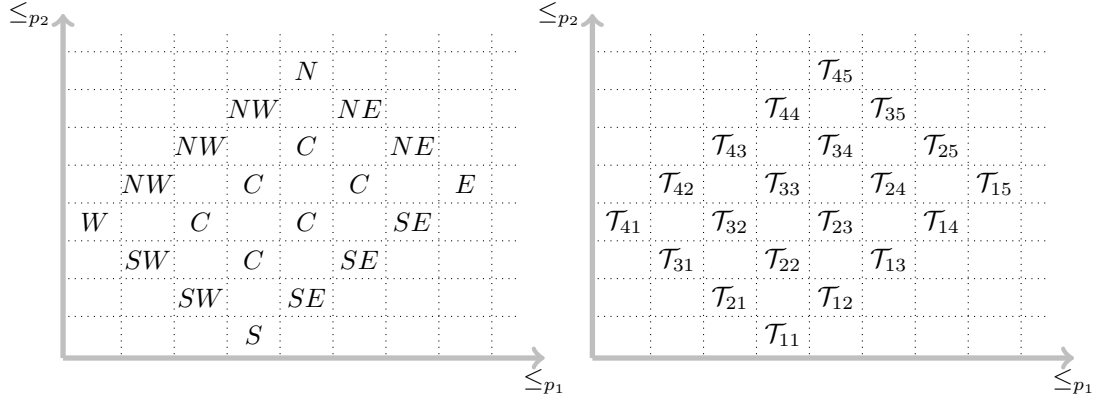
For sake of completeness, we repeat the argument from [1]. We reduce the non-emptiness of multicounter automata to satisfiability of $\text{FO}^2(\leq_{l_1}, +1_{l_1}; +1_{l_2})$. The problem of testing non-emptiness of multicounter automata is known to be equivalent to the reachability problem in vector addition systems.

Let M be a multicounter automaton with counters $C = \{1, \dots, m\}$ and state set Q . Transitions $\Delta = \{\delta_1, \dots, \delta_l\}$ are from the language $Q \times \{D_j \mid j \in C\}^* \times \{I_j \mid j \in C\}^* \times Q$, where D_j and I_j stand for decrementing and incrementing the counter j , respective.

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

- The string projection of the order \leq_{l_1} is of the form $\delta_{i_1} \dots \delta_{i_n}$ such that δ_{i_1} is a transition from the initial state, δ_{i_n} is a transition to a final state and every two successive transitions have a common state.
- The string projection of the order \leq_{l_2} is in the language $Q^*(I_1 D_1 + \dots + I_k D_k)^*$.
- It is the case that $\bigwedge_{j \in C} \forall x \forall y [(I_j(x) \wedge D_j(y) \wedge +1_{l_2}(x, y)) \rightarrow x \leq_{l_1} y]$.

◀



■ **Figure 7** 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.

Proofs from Section 4

Proposition 6. *Finite satisfiability of two-variable logic with two additional preorder successor relations, that is $\text{FO}^2(+1_{p_1}; +1_{p_2})$, 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, \dots, T_k over a set of colors $\{c_1, \dots, c_l\}$.
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 7.

We now describe a $\text{FO}^2(+1_{p_1}; +1_{p_2})$ -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 7 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 .

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 7). 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. ◀

Proposition 7. *Finite satisfiability of two-variable logic with two additional 2-bounded preorder successor relations is undecidable.*

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.

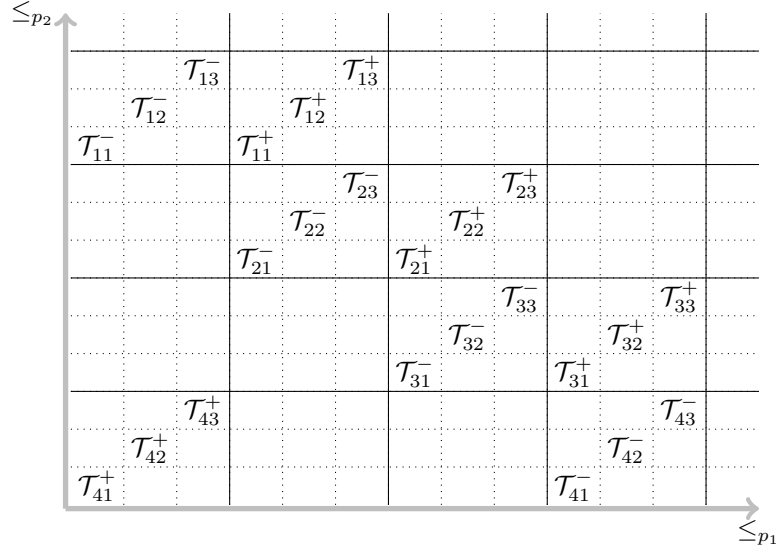
Given a tiling \mathcal{T} of a $m \times n$ grid, we can obtain a $(+1_{p_1}; +1_{p_2})$ -structure $\mathcal{M}(\mathcal{T})$ with only two elements per equivalence class as shown in Figure 8.

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 $r = t_1 \dots t_n$ of a valid tiling is represented by elements $x_1, \dots, x_n, y_1, \dots, y_n$ such that

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



■ **Figure 8** 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.

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} requests for a row completely labeled with tiles are colored with 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^-, \dots, r_n^- be elements from \mathcal{M} such that r_{i+1}^- is northeast of r_i^- for all $i \in \{1, \dots, n-1\}$ and n is maximal. Then the first line of \mathcal{T} contains the tiles t_1, \dots, 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^+, \dots, 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 tiles from C (by (R2)). Condition (N1) guarantees the existence of elements r'_1, \dots, 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 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, \dots, 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. ◀

Corollary 13. *Finite satisfiability of $\text{FO}^2(+1_{l_1}, +2_{l_1}; +1_{l_2}, +2_{l_2})$ 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

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

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

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

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

Proposition 8. *Finite satisfiability of two-variable logic with a linear order, its corresponding successor relation and an additional preorder successor relation, i.e. $\text{FO}^2(+1_l, \leq_l; +1_p)$, is undecidable.*

Proof. The proof follows the lines of the proof of Proposition 29 in [2].

We reduce from the undecidable problem

Problem: PCP

Input: A sequence $(u_1, v_1), \dots, (u_k, v_k)$, where every $u_i, v_i \in \Sigma^*$.

Question: Is there a non-empty, finite sequence $\vec{i} = i_1, \dots, i_m$ such that

$$u_{i_1} \dots u_{i_m} = v_{i_1} \dots v_{i_m}?$$

Let $I = (u_1, v_1), \dots, (u_k, v_k)$ be an instance of PCP. We construct a $\text{FO}^2(\leq_l, +1_l; +1_p)$ -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 is $u_{i_1} v_{i_1} \dots u_{i_m} v_{i_m}$ 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 $+1_p$ 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 $x_1, \dots, x_{|u|}$ corresponding to the positions of $u := u_{i_1} \dots u_{i_m}$ fulfill $+1_p(i, i+1)$ for all $i \in \{1, \dots, |u| - 1\}$. Analogously for v .

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

$$\forall x \forall y (U(x) \wedge U(y) \wedge +1_p(x, y) \rightarrow x <_l y)$$

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