Two Variables and Two Successors

Amaldev Manuel

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

Abstract. We look at the finite satisfiability problem of the two variable fragment of first order logic with the successors of two linear orders. While the logic with both the successors and their transitive closures remains undecidable, we prove that the logic with only the successors is decidable.

1 Introduction

The two variable fragment of the first order logic is known to have an elementarily decidable satisfiability problem [1,2] and reasonably good expressive power (it contains several propositional modal logics). Recently, in the context of verification and semi-structured data, several extensions of FO^2 have been studied [3,4,5]. Strongly inspired by these results, we consider the decidability of FO^2 over finite structures with two linear orders.

A detailed context of the results in this paper is presented later. In the following we give a short account of the results on finite satisfiability problem (henceforth referred as FinSat) of several FO^2 extensions. In [6] it is shown that FinSat of FO^2 over words is Nexptime-complete. [7] shows that FinSat is undecidable for FO^2 with (at least) eight orders. In [8] it is shown that FO^2 with two transitive relations (without equality) is undecidable. In [9] it is shown that FO^2 is undecidable with three equivalence relations, but is decidable when the number of equivalence relations is two. Later in [10] it is shown that in the case of two equivalence relations, FinSat is decidable in 3-Exptime. In the same paper the undecidability is sharpened to one equivalence relation and one transitive relation. In [3] it is shown that FO^2 over words with an equivalence relation is decidable. The same authors in [4] showed that FO^2 over trees with an equivalence relation is decidable when only the successor relation in the tree is used.

Recently, considerations from semi-structured data and infinite state verification motivated the notion of datawords : linear orders labelled by a pair of elements, one from a finite set and one from an infinite domain. Formally, a dataword $w = (a_1, d_1) \dots (a_n, d_n)$, $a_i \in \Sigma$, $d_i \in D$ where Σ is a finite alphabet and D is an infinite set. When operations on D is limited to only equality checking, it is easy to see that w can also be seen as a word with an equivalence relation on its positions. In a landmark paper, the authors of [3] showed that $\mathrm{FO}^2\,(\Sigma, \prec, \prec^+, \sim)$ is decidable. The idea behind the proof, which goes back to

P. Hliněný and A. Kučera (Eds.): MFCS 2010, LNCS 6281, pp. 513–524, 2010.

Büchi, is to define a suitable automaton mechanism (called Data Automata) over datawords and to reduce the formulas of the logic to equivalent automata. Thus, the satisfiability problem of the logic is reduced to a reachability problem of the automaton, which is then shown decidable. In [3] it is also shown that a lot of natural extensions of $\mathrm{FO}^2(\Sigma, \prec, \prec^+, \sim)$ are undecidable, in particular $\mathrm{FO}^2(\Sigma, \prec, \prec^+, \sim, <')$ where $<'$ is a linear order on the data values. When all the data values are distinct, this logic deals with structures with two linear orders.

Almost all the proofs in this paper extensively follow the technique developed in [3]. However, there is a significant departure in the technical details, which we believe will make the results interesting. One crucial aspect where the proof differs is in the use of marking. In the case of Data automata, as shown in [11] the marking can be dispensed with. In the case of text automata it is not clear how to do this. Also, the translation from logic to automata depends on the fact that the marked string projection to the second order can be computed on the fly from the marked string projection to the first order.

Results. Our main result is the following : The finite satisfiability problem of FO^2 over two linear orders where the vocabulary contains only the successor relations of the orders is decidable. Our proof is automata theoretic. We define the notion of a text automaton and study its properties. We show that the emptiness checking of the automaton is decidable in NP. Finally, we show that a formula of the logic can be converted to an equivalent text automaton in 2-Exptime.

Related work. Recently, Thomas Schwentick and Thomas Zeume proved that finite satisfiability of two-variable logic over structures with a linear order and a total preorder $\mathrm{FO}^2(\Sigma, <, \lesssim)$ is Expspace-Complete [12]. Since it is expressible in FO^2 that a total preorder is a linear order, this result implies that FinSat of $\mathrm{FO}^2\left(\Sigma, \prec_1^+, \prec_2^+\right)$ is in Expspace. Their results deal with the order relations and not successor relations, thus renting them incomparable to our results, but certainly complementing the results in this paper. In [13] the author showed that existential MSO over two successors is strictly weaker than over linear orders. (We give an alternative proof of this fact in this paper).

2 Preliminaries

Let T be a finite set. We say a binary relation R on T is a (strict) partial order if it is (1) irreflexive (2) transitive and hence asymmetric. We say a partial order R is a (strict) linear order if it is (3) total. We call the binary relation $S \subseteq R$, defined as $\{(a,b) \mid aRb, \neg\exists c \in T, aRc \wedge cRb\}$ the successor or covering relation of R.

We denote by $\mathbb{N}$ the set of natural numbers and we represent the successor relation on $\mathbb{N}$ by $\prec^{\mathbb{N}}$. For an $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots n\}$. We represent by $\prec_n^{\mathbb{N}}$ the successor relation on $\mathbb{N}$ restricted to the set $[n]$, that is $\prec_n^{\mathbb{N}} = \prec^{\mathbb{N}} \cap ([n] \times [n])$.

Let Σ be an alphabet, a non-empty finite set. A word w over Σ is any finite sequence of characters from Σ. For a word w, we denote the length of w as $|w|$. Given a word $w = a_1 a_2 \dots a_n$ over Σ, we can represent the word as a first order structure $\mathfrak{W} = ([n], \lambda_w, \prec_n^{\mathbb{N}})$, where $\lambda_w : [n] \to \Sigma$ is the labeling function, defined as $\lambda_w(i) = a_i$.

A text $\mathfrak{T}$ over Σ is a first order structure $\mathfrak{T} = (T, \lambda, \prec_1, \prec_2)$ where T is a finite set, $\lambda : T \to \Sigma$ is a labeling function, $\prec_1, \prec_2$ are successor relations of two linear orders over T. We denote the the linear order corresponding to $\prec_1$ (alternatively $\prec_2$) by the symbol $\prec_1^+$ (alternatively $\prec_2^+$). Restricting the structure $\mathfrak{T}$ to either of the orders yields a word, we call the word $(T, \lambda, \prec_1)$ the projection of $\mathfrak{T}$ to the order $\prec_1$.

Given any text $\mathfrak{T} = (T, \lambda, \prec_1, \prec_2)$ where $|T| = n$ we can rewrite $\mathfrak{T}$ uniquely as $([n], \lambda', \prec_n^{\mathbb{N}}, \prec_2')$ such that $\lambda' = \kappa^{-1} \circ \lambda$ and $\prec_2' = \{(\kappa(x), \kappa(y)) \mid x \prec_2 y\}$ where κ is the unique isomorphism from $(T, \prec_1)$ to $([n], \prec_n^{\mathbb{N}})$. Similarly, it can be also rewritten uniquely as $([n], \lambda'', \prec_1', \prec_n^{\mathbb{N}})$.

By $\mathrm{FO}^n(\tau)$ we mean the n-variable first order logic with the vocabulary τ (including equality) over texts. The predicate $\prec_1$ (alt. $\prec_2$) is interpreted as the successor relation in the first (alt. second) linear order. The predicate $\prec_1^+$ (alt. $\prec_2^+$) is interpreted as the first (alt. second) linear order. The predicate $\prec_1^i$ (alt. $\prec_2^i$) is interpreted as the 'i-th successor' relation of the first (alt. second) linear order.

3 Automata on Texts

Given a text of the form $\mathfrak{T} = ([n], \lambda, \prec_1 = \prec_n^{\mathbb{N}}, \prec_2)$, let $([n], \lambda, \prec_n^{\mathbb{N}}) = a_1 a_2 \dots a_n$ be the projection of $\mathfrak{T}$ to the order $\prec_1$. We define the marked string projection of $\mathfrak{T}$ to $\prec_1$, abbreviated as $msp_{\prec_1}(\mathfrak{T})$, as the word $(a_1, b_1)(a_2, b_2) \dots (a_n, b_n)$ where $b_i \in \{-1, 0, 1\}$, such that

$$b_i = \begin{cases} -1 & \text{if } 1 \le i < n \text{ and } i+1 \prec_2 i \\ 1 & \text{if } 1 \le i < n \text{ and } i \prec_2 i+1 \\ 0 & \text{otherwise} \end{cases}$$

Given any text $\mathfrak{T}$ we can define its $msp_{\prec_1}(\mathfrak{T})$ by converting it into the above form.

Similarly, we can define the marked string projection of $\mathfrak{T}$ to $\prec_2$ denoted as $msp_{\prec_2}(\mathfrak{T})$. For this, we first convert it into the form $\mathfrak{T}' = ([n], \lambda, \prec_1, \prec_2 = \prec_n^{\mathbb{N}})$ where $|T| = n$. let $([n], \lambda, \prec_n^{\mathbb{N}}) = a_1 a_2 \dots a_n$ be the projection of $\mathfrak{T}'$ to the order $\prec_2$. $msp_{\prec_2}(\mathfrak{T}) = msp_{\prec_2}(\mathfrak{T}')$ is defined as the word $(a_1, b_1)(a_2, b_2) \dots (a_n, b_n)$ where $b_i \in \{-1, 0, 1\}$, such that

$$b_i = \begin{cases} -1 & \text{if } 1 \le i < n \text{ and } i+1 \prec_1 i \\ 1 & \text{if } 1 \le i < n \text{ and } i \prec_1 i+1 \\ 0 & \text{otherwise} \end{cases}$$

In the following we define the notion of a text automaton. Fix an alphabet Σ. A text automaton $A = (B, C)$ is a composite automaton consisting of two word

automata B and C. The automaton B is a non-deterministic letter-to-letter word transducer with the input alphabet $\Sigma \times \{-1, 0, 1\}$ and an output alphabet Σ' (included in the definition of B). The automaton C is a non-deterministic finite state recognizer accepting words over the alphabet Σ'. Given a text $\mathfrak{T} = (T, \lambda, \prec_1, \prec_2)$ the automaton works as follows. The transducer B runs over the $msp_{\prec_1}(\mathfrak{T})$ yielding a string $w = (T, \lambda', \prec_1)$ in Σ'^*, where $\lambda' : T \to \Sigma'$. The automaton C runs over the string $w' = (T, \lambda', \prec_2)$, notice that w is permuted to the order $\prec_2$. Finally, the automaton A accepts $\mathfrak{T}$ if both B and C have a successful run, that is they both finish in one of their final states respectively.

Formally, let $A = (B, C)$, where B is a word transducer given by the tuple $B = (Q_b, (\Sigma \times \{-1, 0, 1\}), \Sigma', O_b, \Delta_b, I_b, F_b)$, where Q_b is the finite set of states, $(\Sigma \times \{-1, 0, 1\})$ is the input alphabet, Σ' is the output alphabet, $I_b \subseteq Q_b$ is the set of initial states, $F_b \subseteq Q_b$ is the set of final states, $\Delta_b \subseteq Q_b \times (\Sigma \times \{-1, 0, 1\}) \times Q_b$ is the set of transitions and $O_b : Q_b \times (\Sigma \times \{-1, 0, 1\}) \to \Sigma'$ is the output function. Given a marked string $w = (a_1, b_1)(a_2, b_2) \dots (a_n, b_n)$ we define a run ρ_B of B as a sequence $q_0 q_1 \dots q_n$ such that $q_0 \in I_b$ and for every $i \in [n]$ there is a transition $(p, (a, b), q)$ in Δ_b such that $q_{i-1} = p$, $q_i = q$, $a_i = a$ and $b_i = b$. The run ρ_b is accepting if $q_n \in F_b$. Given an accepting run ρ_b of B on w, it uniquely defines an output string $w' = a'_1 a'_2 \dots a'_n \in \Sigma'^*$ where $a'_i = O_b(q_{i-1}, a_i)$. The automaton C is given by the tuple $C = (Q_c, \Sigma', \Delta_c, I_c, F_c)$ where Q_c is the finite set of states, Σ' is the alphabet, $I_c \subseteq Q_c$ is the set of initial states, $F_c \subseteq Q_c$ is the set of final states, $\Delta_c \subseteq Q_c \times \Sigma' \times Q_c$ is the set of transitions. Given a word $w' = a'_1 a'_2 \dots a'_n \in \Sigma'^*$ a run ρ_c of C is a sequence $q_0 q_1 \dots q_n$ such that $q_0 \in I_c$ and for every $i \in [n]$ there is a transition (p, a', q) in Δ_c such that $q_{i-1} = p$, $q_i = q$ and $a'_i = a'$. The run ρ_c is accepting if $q_n \in F_c$. Now, we define the run ρ of A on the text $\mathfrak{T} = (T, \lambda, \prec_1, \prec_2)$ as a pair (ρ_b, ρ_c) such that (i) ρ_b is an accepting run of B on $msp_{\prec_1}(\mathfrak{T})$ yielding a word $(T, \lambda', \prec_1)$ and (ii) ρ_c is an accepting run of C on the word $(T, \lambda', \prec_2)$.

We look at some example languages. Let L_1 be the language of texts where both the orders coincide, that is $L_1 = \{\mathfrak{T} \mid \mathfrak{T} \models \forall xy\ x \prec_1^+ y \Leftrightarrow x \prec_2^+ y\}$. This is easily done by checking the markings. What is more interesting is that we can accept texts whose string projections are non-regular. Consider the formula, $\varphi = \varphi_{\prec_1} \wedge \varphi_{\prec_2}$, where $\varphi_{\prec_1}$ says that the word projection of the text to the order $\prec_1$ belongs to the language $a^*b^*c^*$, whereas $\varphi_{\prec_2}$ says that the projection to $\prec_2$ belongs to the language $(abc)^*$. Hence, the projection of the text to $\prec_1$ has to be the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$. A text automaton can accept $L(\varphi)$ in the following way, the transducer projects the marked string to Σ and checks if it belongs to $a^*b^*c^*$. The automaton C checks if its input is in $(abc)^*$.

Lemma 1. *Given a regular language $L \subseteq \Sigma^*$, there is a text automaton accepting all texts whose projections to $\prec_1$ is in L. Similarly, there is a text automaton accepting all texts whose projections to $\prec_2$ is in L.*

Proof. In the first case the transducer B checks if the projection of the text to $\prec_1$ (ignoring the markings) is in L and C accepts Σ'^*. For the second case, the transducer B simply copies the string (again ignoring the markings) and C accepts if its input is in L.

Lemma 2. *Languages recognized by text automata are closed under union, intersection and renaming.*

Proof. Closure under union and intersection is obtained from usual product construction (using a composed output alphabet). Closure under renaming is achieved using the non-determinism of the transducer.

Consider the language L_M, the collection of texts such that (1) the projection to $\prec_1$ is in $\$_1 a^+ \#_1 \$_2 b^+ \#_2$. (2) projection to $\prec_2$ is in $\$_1\$_2(ab)^+\#_1\#_2$. (3) There exists two positions x_0, x_1 having the same label from $\{a, b\}$ such that $x_0 \prec_1^+ x_1$ and $x_1 \prec_2^+ x_0$. The language, L_M is accepted by a text automaton. Conditions 1 and 2 can be checked easily by B and C. For condition 3, the transducer B non-deterministically chooses two positions having the same label (either a or b), $x_0 \prec_1^+ x_1$ and outputs 0 at x_0 and 1 at x_1 and \$ at every other position. The automaton C verifies that its input is of the form $\$^*1\$^*0\$^*$.

But $\overline{L_M}$, the complement of the above language is not accepted by any text automaton. For the sake of contradiction, assume that there is a text automaton $A = (B, C)$ accepting $\overline{L_M}$. Let the number of states in B be n. Consider the text $\mathfrak{T}_1 = \left([2k+4], \lambda, \prec_{2k+4}^{\mathbb{N}}, \prec_2\right)$ such that $\left([2k+4], \lambda, \prec_{2k+4}^{\mathbb{N}}\right)$ is $\$_1 a^k \#_1 \$_2 b^k \#_2$ and $\prec_2$ is the order $\{(1, k+3), (k+3, 2), (2, k+4), (k+4, 3) \ldots (k+2, 2k+4)\}$ where $k > n$. Note that in the $\text{msp}_{\prec_1}(\mathfrak{T})$ all the markings are zero. Since $\mathfrak{T}$ is in $\overline{L_M}$, there is an accepting run of B such that there exists two positions $i < j$ with label a and $q_{i-1} = q_{j-1}$ in the run. We define the order $\prec_2'$ as

$$\begin{aligned}\{(l, k+2+l) \mid 1 \le l \le k+2, l \ne i, l \ne j\} \\ \cup \{(k+2+l, l+1) \mid 1 \le l < k+2, l+1 \ne i, l+1 \ne j\} \\ \cup \{(k+1+i, j), (j, k+2+i), (k+1+j, i), (i, k+2+j)\}\end{aligned}$$

In the relation $\prec_2'$ only the positions i and j are switched from $\prec_2$. Let $\mathfrak{T}_1' = \left([2k+4], \lambda, \prec_{2k+4}^{\mathbb{N}}, \prec_2'\right)$. It is the case that $\text{msp}_{\prec_1}(\mathfrak{T}) = \text{msp}_{\prec_1}(\mathfrak{T}')$ and B has an accepting run on $\text{msp}_{\prec_1}(\mathfrak{T}')$ outputting the same string as in the case of $\mathfrak{T}$, which then permuted to $\prec_2$ and $\prec_2'$ gives the same string. Hence C also has an accepting run. But, $\mathfrak{T}'$ does not belong to $\overline{L_M}$, leading to a contradiction. This shows that,

Lemma 3. *The class of languages accepted by text automata are not closed under complementation.*

Using a similar argument, we can show that the class of text automata where the transducer B is deterministic is strictly weaker.

We can prove the following proposition analogous to the $\text{EMSO}^2(\Sigma, \prec)$ characterisation (Büchi–Elgot–Trakhtenbrot) of finite state automata.

Proposition 1. *For every text automaton A, there is an* $\text{EMSO}^2(\Sigma, \prec_1, \prec_2)$ *formula φ_A such that $L(A) = L(\varphi_A)$.*

4 Decidability of the Text Automaton

In the definition of the automaton A, the transducer has access to $\text{msp}_{\prec_1}(\mathfrak{T})$, whereas C can only access the output of B permuted to $\prec_2$. In the following, we

show that it is possible for B to output a string which then permuted to $\prec_2$, yields $\mathrm{msp}_{\prec_2}(\mathfrak{T})$. Let $\mathfrak{T} = (T, \lambda, \prec_1, \prec_2)$ be a text and $\mathrm{msp}_{\prec_1}(\mathfrak{T}) = (a_1, b_1)(a_2, b_2)\dots(a_n, b_n)$. Let b'_i be the marking of the position i in $\mathrm{msp}_{\prec_2}(\mathfrak{T})$. It is easy to verify that b'_i is a function of b_i and b_{i-1} as evidenced by the following table.

b_{i-1}	b_i	b'_i	b_{i-1}	b_i	b'_i	b_{i-1}	b_i	b'_i
$-$	0	0	0	1	1	-1	0	-1
$-$	-1	0	1	0	0	-1	-1	-1
$-$	1	1	1	1	1	-1	1	$\perp$
0	0	0	0	-1	0	1	-1	$\perp$

Note that the configurations $b_{i-1} = -1$, $b_i = 1$ and $b_{i-1} = 1$, $b_i = -1$ does not constitute a valid marking. The above table immediately gives a strategy for outputting $\mathrm{msp}_{\prec_2}(\mathfrak{T})$. The automaton B always remembers the previous position's marking in its states, and computes b'_i. Once the output of B is permuted to $\prec_2$, the string becomes $\mathrm{msp}_{\prec_2}(\mathfrak{T})$.

Using the above fact, we can show that the emptiness problem for a text automaton is decidable in NPTIME. The idea is the following: Given a text automaton $A = (B, C)$, $L(A)$ is non-empty if there is a marked word w accepted by B such that a permutation of output of B on w, 'consistent' with the marking of w, is accepted by C.

Fix an alphabet Σ. Given a marked word $w \in (\Sigma \times \{0, 1, -1\})^*$ we denote the projection of w to Σ by $w \downarrow \Sigma$. Let $w = ([|w|], \lambda, \prec^{\mathbb{N}})$ be a marked word. A permutation $\pi : [|w|] \to [|w|]$ is a bijection and $\pi(w)$ is defined as the word $\left([|w|], \pi^{-1} \circ \lambda, \prec^{\mathbb{N}}_{|w|}\right)$. Note that π defines an order $\prec_\pi = \pi^{-1}(1) \dots \pi^{-1}(|w|)$ on the positions of w. We say that the permutation π is consistent with the marking if w is the marked string projection of the text $\mathfrak{T} = \left([|w|], \lambda, \prec^{\mathbb{N}}_{|w|}, \prec_\pi\right)$ to the order $\prec^{\mathbb{N}}_{|w|}$. Given a word $w' \in \Sigma^*$, by $\mathrm{mperm}(w')$, we denote the set of all the marked words w such that there is a permutation π consistent with the marking such that $\pi(w \downarrow \Sigma) = w'$. Given $L \subseteq \Sigma^*$, we define $\mathrm{mperm}(L) = \cup_{w' \in L} \mathrm{mperm}(w')$. Next we show that if L is regular then $\mathrm{mperm}(L)$ is accepted by a Presburger automaton.

A Presburger automaton A is a tuple (B, φ) where B is a finite state automaton with states $q_0, \dots q_n$ and φ is a Presburger formula with free variables $|q_0|, \dots, |q_n|$. A word w is accepted by the automaton A if there is an accepting run ρ of B on w such that $\varphi(|q_0|, \dots, |q_n|)$ is true, where $|q_i|$ is the number of times q_i appears in ρ.

Lemma 4. *If L is regular then* $\mathrm{mperm}(L)$ *is accepted by a Presburger automaton.*

Proof. The idea is to adapt the technique from [14]. Let $A = (Q, \Sigma, \Delta, q_0, F)$ be a non-deterministic finite state automaton accepting L. Given a marked word $w = (a_0, b_0) \dots (a_n, b_n) \in (\Sigma \times \{0, -1, 1\})^*$, the Presburger automaton P_A checks non-deterministically if there is a run of A on some consistent permutation of w. To achieve this, the automaton P_A assigns a transition $\delta = (p, a_i, q) \in \Delta$

to each position i of the marked word. The information whether the transition should or should not be 'joined' with the transitions of the neighbouring positions is also remembered. All this information is stored in the state occuring immediately after position i. If the marking is 1 or -1 the automaton ensures that the successive transitions associated with the positions are consistent with the marking. We can define a flow f where each transition δ of A is labelled by the number of times it is associated with a position. Finally, we can write a Presburger formula which checks that,

1. f is locally consistent.
2. the subgraph induced by the states with a non-zero flow is connected.
3. f is consistent with the marking.

Properties 1 and 2 is encoded as in the proof of theorem 1 in [14]. Property 3 is easily encoded as a summation. The resulting automata P_A is poly-sized in terms of the size of A.

Theorem 1. *Emptiness checking of a text automaton is in* NP.

Proof. Given the text automaton $A = (B, C)$, we construct a Presburger automaton P_C which accepts $\mathrm{mperm}(L(C))$. We take the intersection of the transducer B and P_C such a way that the output of B is supplied as the input of P_C. Finally we check the emptiness of the resulting automaton which is in NP.

Next we show that the emptiness problem of Presburger automata which is NP-hard reduces to the emptiness problem for text automata, yielding a lower bound.

Theorem 2. *Emptiness checking of a Presburger automaton is polynomial time reducible to the emptiness checking of a text automaton.*

Proof. We use an alternative definition of Presburger automaton from [5], which is a type of automata with counters. A Presburger automaton P_A is a finite state automaton with a set of counters $\{1 \ldots k\}$ holding integer (possibly negative) values. Formally it is a tuple $(Q, \Sigma, \Delta, q_0, F)$ where Q is the finite set of states, q_0 is the initial state and F is the set of final states. The transition relation Δ is a subset of $Q \times \Sigma \times \{I(j), D(j) \mid 1 \leq j \leq k\} \times Q$. A configuration of the automaton is of the form $(p, \bar{u})$ where $p \in Q$ and $\bar{u} : [k] \to \mathbb{Z}$. The automaton starts in the initial state with all the counters being zero. On a state p with counter values $\bar{u}$, the automaton can make a transition $(p, a, I(j), q)$ (alt. $(p, a, D(j), q)$) on the letter a resulting in the state q with counter values being the same, except for the counter j which is incremented (alt. decremented) by one. Finally at the end of the word, the automaton accepts if it reaches an accepting state with all the counters being zero.

For emptiness checking, without loss of generality we can ignore the labels on the transitions. We construct a text automaton whose alphabet is $Q \cup \{I_1, D_1, \ldots, I_k, D_k\}$. We can represent the transition $(p, I(j), q)$ as the word pI_jq over this alphabet. A run ρ of P_A is represented by a text where (1) the string projection

to the first order represents a sequence of transitions $\delta_1 \dots \delta_n$, $\delta_i \in \Delta$ which is consistent — the run starts in the initial state, ends in a final state and two successive transitions have a common state (2) the string projection to the second order belongs to the language $Q^* \cdot (I_1 D_1 + \ldots + I_k D_k)^*$. This language can be recognized using a text automaton $A = (B, C)$ where B verifies (1) and C verifies (2). It is easy to see that $L(P_A)$ is non-empty if and only if $L(A)$ is non-empty.

5 Reduction from Logic to Automata

In this section we show that given an $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ formula we can transform it into an equivalent text automaton. Below, we shorthand the formula $\neg x \prec_1 y$ as $x \not\prec_1 y$, similarly the others too. First of all, given a formula $\varphi \in \mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ we transform it into an equivalent formula in *Scott normal form* which is of the form,

$$\exists R_1 \dots R_n \left(\forall x \forall y \ \chi \wedge \bigwedge_i \forall x \exists y \ \psi_i \right)$$

where the predicates R_i are unary, and χ and ψ_i are quantifier-free formulas in $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$. The resulting formula is linear in terms of the size of the original formula. Earlier we showed that text automata are closed under renaming and intersection. Therefore it suffices to show that we can construct a text automaton for each of the formulas $\forall x \exists y \ \chi$ and $\forall x \exists y \ \psi_i$. The following two lemmas show precisely that.

In the following a *type* is a one variable quantifier-free formula containing only unary predicates.

Lemma 5. *Given an* $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ *formula of the form* $\varphi = \forall x \forall y \ \chi$ *where* χ *is quantifier free, an equivalent text automaton of doubly exponential size can be constructed.*

Proof. Firstly, we write φ in CNF causing an exponential blowup in the size of the formula, followed by distributing the universal quantification over the conjunctions and rewriting the formula as $\bigwedge_i \forall x \forall y \ \chi_i$ where each χ_i is of the form,

$$\chi_i = \alpha(x) \vee \beta(y) \vee \epsilon(x, y) \vee \delta_1(x, y) \vee \delta_2(x, y).$$

Above $\alpha(x)$ and $\beta(y)$ are unary types. The formulas in the group $\epsilon(x, y)$ are $x = y$ and $x \neq y$. The formula δ_1 is a disjunction of literals from the set O_1 and δ_2 is a disjunction of literals from the set O_2, where $O_1 = \{x \prec_1 y, x \not\prec_1 y, y \prec_1 x, y \not\prec_1 x\}$ and $O_2 = \{x \prec_2 y, x \not\prec_2 y, y \prec_2 x, y \not\prec_2 x\}$. It is enough to construct a text automaton for each χ_i since the automata are closed under intersection. The alphabet Σ of the automata is going to be bit vectors which represent the evaluation of the unary predicates including R_i used in the formula at a given position. Hence, the size of the alphabet is exponential in the length of the formula. The automaton

we construct in each case has constant number of states, but may have exponentially many transitions. Finally the intersection of these automata is of size doubly exponential.

We note that whenever χ_i describes a regular property, we can construct an equivalent text automaton by converting the finite state automaton equivalent to χ_i. If one of $\delta_1(x,y)$ and $\delta_2(x,y)$ is absent, the formula χ_i describes a regular property over one linear order. Therefore we restrict our attention to those χ_i where both δ_1 and δ_2 are present. Suppose $\epsilon(x,y) \equiv x = y \vee x \neq y$. In this case, the formula is tautology hence we construct a text automaton which accepts all texts.

Suppose $\epsilon(x,y) \equiv x \neq y$. In this case we can rewrite χ_i as,

$$\chi_i \equiv [\alpha'(x) \wedge \beta'(y) \wedge x = y] \rightarrow [\delta_1(x,y) \vee \delta_2(x,y)]\,.$$

Substituting x for y, we can see that the consequent reduces to either $\top$ or $\bot$. In both cases, the formula describes a regular property.

When $\epsilon(x,y) \equiv x = y$ and $\delta_1(x,y)$ or $\delta_2(x,y)$ contains a negative literal, we can rewrite χ_i in one of the following two forms,

$$\chi_i \equiv [\alpha'(x) \wedge \beta'(y) \wedge x \neq y \wedge \delta_1'(x,y)] \rightarrow \delta_2(x,y),$$
$$\chi_i \equiv [\alpha'(x) \wedge \beta'(y) \wedge x \neq y \wedge \delta_2'(x,y)] \rightarrow \delta_1(x,y),$$

where $\alpha', \beta', \delta_1', \delta_2'$ are the negations of $\alpha, \beta, \delta_1, \delta_2$ respectively. We choose the appropriate form such that $\delta_1'(x,y)$ or $\delta_2'(x,y)$ contains a positive literal. In this case the automaton can verify the formula χ_i by looking at the marked string projection to $\prec_1$ or $\prec_2$ depending upon the chosen form.

The only remaining case is when $\epsilon(x,y) \equiv x = y$, and neither δ_1 nor δ_2 contains a negative literal, that is when δ_1 and δ_2 are disjunctions of positive literals. We rewrite χ_i in the following form,

$$\chi_i = [\alpha'(x) \wedge \beta'(y) \wedge x \neq y] \rightarrow [\delta_1(x,y) \vee \delta_2(x,y)]\,.$$

The formula says the following. Whenever α' holds at x and β' holds at y and x, y are distinct then either they are neighbours in $\prec_1$ as dictated by δ_1 or neighbours in $\prec_2$ as dictated by δ_2. If there is no α' in the word there can be any number of β'. Similarly there can be any number of α' if there is no β' occurring in the word. The automaton B can guess both these cases and verify them easily. When there is at least one α' and β' present in the word the number of α' and β' are bounded. Therefore in this case the formula χ_i can be checked by a text automaton by labelling the α' and β'.

This completes the proof.

Lemma 6. *For each* $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ *formula of the form* $\forall x \exists y\ \psi$ *where* ψ *is quantifier free, an equivalent text automaton of doubly exponential size can be constructed.*

Proof. First of all, ψ can be written (using the truth table for ψ) as an exponential size conjunction of disjunctions of the form $\forall x \exists y\ \bigwedge_i \bigvee_j [\alpha_i(x) \rightarrow \theta_{ij}(x,y)]$,

where α_i enumerates through all possible maximal types, that is $\bigvee_i(\alpha_i(x))$ is a tautology and $\neg(\alpha_i(x) \wedge \alpha_j(x))$ for all $i \neq j$. The formula θ_{ij} is either $\perp$ or of the form,

$$\beta(y) \wedge \epsilon(x,y) \wedge \delta_1(x,y) \wedge \delta_2(x,y)$$

where, β is a type, ϵ is one of $x = y$, $x \neq y$, δ_1 is in O_1 and δ_2 is in O_2.

We notice that the premise occurring in distinct conjuncts are distinct (and mutually exclusive). Hence it is possible to distribute the $\forall x \exists y$ over the conjunction. The resulting formula is of the form, $\wedge_i \forall x \exists y \ \vee_j (\alpha_i(x) \supset \theta_{ij}(x,y))$. We eliminate the disjunction by adding to every disjunct a new unary predicate $\Lambda_{ij}(x)$ which denotes that at the position x, the j-th disjunct is witnessing α_i. We can rewrite every conjunct in the above formula as,

$$\exists \Lambda_{i1} \Lambda_{i2} \ldots (\forall x \vee_j \Lambda_{ij}(x)) \wedge \bigwedge_j \forall x \exists y \ [(\alpha_i(x) \wedge \Lambda_{ij}(x)) \rightarrow \theta_{ij}(x,y)]$$

A text automaton can guess the predicates Λ_{ij}. So it is enough to construct a text automaton for each formula of the form $\forall x \exists y \ [\alpha(x) \supset \theta_{ij}(x,y)]$. If the consequent is false, the language is regular. So we concentrate on the cases where the consequent is satisfiable.

$$\forall x \exists y \ [\alpha(x) \rightarrow (\beta(y) \wedge \epsilon(x,y) \wedge \delta_1(x,y) \wedge \delta_2(x,y))]$$

We do a case analysis. If $\epsilon(x,y) \equiv x = y$, the language is regular. Hence now onwards we fix ϵ to be $x \neq y$.

As in the previous proof, we have two cases, when δ_1 or δ_2 contains a positive literal and when they do not. If δ_1 or δ_2 contains a positive literal, we can easily verify the formula by looking at the marked string projection to the appropriate order.

The only remaining case is when neither δ_1 nor δ_2 contains a positive literal. Consider the case when $\delta_1 \equiv x \not<_1 y$ and $\delta_2 \equiv x \not<_2 y$. The formula says that if there is an α at x there should be a witness y with β holding there, such that y is not a successor of x in both the orders. Notice that if there are at least four β occurring in the word we will be able to find a witness for any α. The automaton guesses whether the word contains at least four β and verifies it, in which case the formula is taken care of. If the automaton guesses that the word contains fewer than four β, it labels each β distinctly and verifies that for every αthere is atleast one β witnessing it. Since the number of β is bounded, this can be done easily.

In the cases where $\delta_1 \wedge \delta_2$ is one of $y \not<_1 x \wedge x \not<_2 y$, $x \not<_1 y \wedge y \not<_2 x$, $y \not<_1 x \wedge y \not<_2 x$, the sufficient number of β is three. When $\delta_1 \wedge \delta_2$ is $x \not<_1 y \wedge y \not<_1 x \wedge x \not<_2 y \wedge y \not<_2 x$, the sufficient number of β is five. In all other cases the sufficient number of β is four. In all the above cases, the construction is similar.

This completes the proof.

Now, we can state the main result.

Theorem 3. FinSat *of* $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ *is in* 2-Nexptime.

In [15], it is shown that FinSat of $\mathrm{FO}^2(\Sigma)$ and FinSat of $\mathrm{FO}^2(\prec)$ with one unary predicate are Nexptime-hard. This lower bound applies to FinSat of $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ as well.

Using a reduction to PCP similar to the one in [3] we can show the following.

Theorem 4. *The satisfiability problems for the following logics are undecidable.*

(a) $\mathrm{FO}^2\left(\Sigma, \prec_1, \prec_1^+, \prec_2, \prec_2^+\right)$
(b) $\mathrm{FO}^3\left(\Sigma, \prec_1, \prec_2\right)$
(c) $\mathrm{FO}^2\left(\Sigma, \prec_1, \prec_1^2, \prec_1^3, \prec_2, \prec_2^2\right)$

The above theorem has to be compared with Proposition 29 from [3].

6 Discussion and Conclusion

In this paper we introduced a class of automata working over texts. Using this we proved that $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ is decidable in 2-Nexptime. The present lower bound is the obvious one, Nexptime, leaving a gap of one exponent.

In the beginning we showed that $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2, \prec_1^+, \prec_2^+)$ is undecidable. From [12] we know that $\mathrm{FO}^2(\Sigma, \prec_1^+, \prec_2^+)$ is decidable in Expspace. This leaves open the decidability question of the following fragments (we omit the symmetric cases) $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2^+)$, $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2, \prec_2^+)$. However, it is interesting to note that these fragments contain languages which are not accepted by any text automaton. Consider the following languages,

$$L_1 = \{\mathfrak{T} \mid \mathfrak{T} \models \forall x \forall y \ \left(a(x) \wedge a(y) \wedge x \prec_1^+ y \supset x \prec_2^+ y\right)\}$$

$$L_2 = \{\mathfrak{T} \mid \mathfrak{T} \models \forall x \forall y \ \left(a(x) \wedge a(y) \wedge x \prec_1 y \supset x \prec_2^+ y\right)\}$$

The language L_1 is definable in $\mathrm{FO}^2(\Sigma, \prec_1^+, \prec_2^+)$ and L_2 in $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2^+)$ [also in $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2, \prec_2^+)$]. To show that these languages are not accepted by any text automaton, we refer back to lemma 3. We considered a text $\mathfrak{T}_1$ which belongs to $\overline{L_M}$, note that $\mathfrak{T}_1$ also belongs to both L_1 and L_2. Using a similar argument we can construct the text $\mathfrak{T}_1'$ which does not belong to L_1 (also L_2), but is accepted by any automata accepting L_1 (alternatively L_2). This essentially shows that our automaton is unable to check any transitive relations. This also implies the theorem which is proved in [13].

We want to draw the attention to why the decidability proof does not generalize to $\mathrm{FO}^2\left(\Sigma, \prec_1, \prec_2, \prec_3\right)$. The reason is that we relied on $\mathrm{msp}_{\prec_1}$ to compute $\mathrm{msp}_{\prec_2}$. This step is not possible in the case of three successor relations. This can be however overcome by providing the component automata (each running on $\prec_1, \prec_2$ and $\prec_3$) with their own marked string projection, in which case it is not clear how to prove the decidability of the text automaton.

Acknowledgements. We wish to emphasize that a suggestion by Luc Segoufin — to use Presburger Arithmetic to show the decidability — has helped to greatly improve the complexity results in this paper. The previous version of this paper used Multicounter Automata for showing the decidability, yielding no elementary complexity bounds. We thank him for the invaluable discussions and suggestions. We also thank Thomas Schwentick and Thomas Zeume for sharing their results with us and in particular Thomas Schwentick for very informative discussions and pointing out an error in the previous version.

References

1. Mortimer, M.: On languages with two variables. Zeitschr. f. Logik und Grundlagen d. Math. 21, 135–140 (1975)
2. Grädel, E., Kolaitis, P.G., Vardi, M.Y.: On the decision problem for two-variable first-order logic. Bulletin of Symbolic Logic 3(1), 53–69 (1997)
3. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: LICS, pp. 7–16 (2006)
4. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. J. ACM 56(3) (2009)
5. Björklund, H., Bojanczyk, M.: Shuffle expressions and words with nested data. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 750–761. Springer, Heidelberg (2007)
6. Etessami, K., Vardi, M.Y., Wilke, T.: First-order logic with two variables and unary temporal logic. In: LICS, pp. 228–235 (1997)
7. Otto, M.: Two variable first-order logic over ordered domains. J. Symb. Log. 66(2), 685–702 (2001)
8. Kieronski, E.: Results on the guarded fragment with equivalence or transitive relations. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 309–324. Springer, Heidelberg (2005)
9. Kieronski, E., Otto, M.: Small substructures and decidability issues for first-order logic with two variables. In: LICS, pp. 448–457. IEEE Computer Society, Los Alamitos (2005)
10. Kieronski, E., Tendera, L.: On finite satisfiability of two-variable first-order logic with equivalence relations. In: LICS, pp. 123–132. IEEE Computer Society, Los Alamitos (2009)
11. Björklund, H., Schwentick, T.: On notions of regularity for data languages. Theor. Comput. Sci. 411(4-5), 702–715 (2010)
12. Schwentick, T., Zeume, T.: Two-variable logic with two order relations. In: Dawar, A., Veith, H. (eds.) CSL 2010 (to appear 2010)
13. Mathissen, C.: Existential mso over two successors is strictly weaker than over linear orders. Theor. Comput. Sci. 410(38-40), 3982–3987 (2009)
14. Seidl, H., Schwentick, T., Muscholl, A., Habermehl, P.: Counting in trees for free. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 1136–1149. Springer, Heidelberg (2004)
15. Etessami, K., Vardi, M.Y., Wilke, T.: First-order logic with two variables and unary temporal logic. Inf. Comput. 179(2), 279–295 (2002)