Register Automata with Extrema Constraints, and an Application to Two-Variable Logic*

Szymon Toruńczyk
University of Warsaw
szymtor@mimuw.edu.pl

Thomas Zeume
Ruhr-Universität Bochum
thomas.zeume@rub.de

Abstract

We introduce a model of register automata over infinite trees with extrema constraints. Such an automaton can store elements of a linearly ordered domain in its registers, and can compare those values to the suprema and infima of register values in subtrees. We show that the emptiness problem for these automata is decidable.

As an application, we prove decidability of the countable satisfiability problem for two-variable logic in the presence of a tree order, a linear order, and arbitrary atoms that are MSO definable from the tree order. As a consequence, the satisfiability problem for two-variable logic with arbitrary predicates, two of them interpreted by linear orders, is decidable.

CCS Concepts • **Theory of computation → Finite Model Theory**; **Tree languages**; **Automata over infinite objects**.

Keywords register automata, two-variable logic, decidability

ACM Reference Format:
Szymon Toruńczyk and Thomas Zeume. 2020. Register Automata with Extrema Constraints, and an Application to Two-Variable Logic. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3373718.3394748

1 Introduction

Automata for words and trees find applications in diverse areas such as logic, verification, and database theory (see, e.g., [1, 15, 23]). Applications to logic include proofs of decidability of the satisfiability problem for various logics, and this is the theme of this paper. Many variations of automata for specific applications have been introduced, among them automata over infinite words or trees, with output, timed automata, or automata working on words and trees whose positions are annotated by data from an infinite domain. In this article we study a variant of the latter family of automata called *register automata* [9, 16].

In its basic form, a register automaton extends a finite state automaton by registers which can store data values from an infinite domain $\mathbf{D}$. The inputs are *data words*, i.e., words labeled by pairs consisting of a label from a finite alphabet Σ, and a data value from $\mathbf{D}$. When reading a data word, a register automaton can store values from $\mathbf{D}$ in its registers. Its state depends on the previous state, the label at the current position as well as the relationship of the stored register values to the data value at the current position. Here, depending on the automaton model at hand, register values can be tested for equality, compared with respect to some linear order on $\mathbf{D}$, or others.

In this article we study a variant of register automata for infinite *data trees*, where the data values form a complete dense total order. In addition to the ability of comparing data values according to the linear order of $\mathbf{D}$, our automaton model allows to compare register values to the suprema and infima over values of registers in a subtree.

We show that the emptiness problem for this automaton model can be solved algorithmically.

Theorem 1.1. *The emptiness problem for tree register automata with suprema and infima constraints is decidable.*

As an application of the above result, we consider the satisfiability problem for variants of two-variable logic. In two-variable first-order logic (short: FO^2) formulas may use only two variables x and y which can be reused. The extension by existential second-order quantifiers is denoted by ESO^2, or EMSO^2 if only monadic such quantifiers are allowed. Two-variable first-order logic is reasonably expressive and enjoys a decidable satisfiability problem [14, 21]. However, an easy application of a two-pebble Ehrenfeucht-Fraïsse game yields that FO^2 cannot express transitivity of a binary relation. For this reason, FO^2 has been studied on structures where some special relations are required to be transitive.

*The work of S. Toruńczyk was supported by Poland's National Science Centre grant 2018/30/E/ST6/00042.

LICS '20, July 8–11, 2020, Saarbrücken, Germany

ACM ISBN 978-1-4503-7104-9/20/07…$15.00
https://doi.org/10.1145/3373718.3394748

A particular interest has been in deciding the satisfiability problem for such extensions. Recently a decision procedure for the finite satisfiability problem for ESO^2 with one transitive relation and for ESO^2 with one partial order have been obtained [18]. Previously ESO^2 with two equivalence relations [11, 12] and ESO^2 with two "forest" relations have been shown to be decidable [4]. While it is known that EMSO^2 with three equivalence relations is undecidable, this problem is still open for three "forest" relations. For ESO^2 with two linear orders, only a decision procedure for the finite satisfiability problem was known [20, 24]. The satisfiability problem and the finite satisfiability problem for EMSO^2 is undecidable for three linear orders [10].

The question whether two-variable logic with two linear orders is decidable for general (not necessarily finite) structures has been left open in [20, 24], and is settled here affirmatively. Beyond settling the question itself, we believe that the techniques developed here might also be interesting in their own rights and applied to other problems, much like in the case of finite satisfiability, where the used techniques were later exploited in work by Dartois, Filiot, and Lhote on transducers [5] and in recent work by Lutz, Jung, and Zeume in relation to description logics [8].

In fact, we prove a more general result. A partial order $(\mathbf{D}, <)$ is a *tree order* (also called a *semi-linear order*) if the set $\{y \mid y < x\}$ is totally ordered by $<$ for each $x \in \mathbf{D}$, and any two elements x, y have some lower bound.

Theorem 1.2. *Countable satisfiability of* ESO^2 *with one tree order, one linear order, and access to MSO-defined atoms over the tree order, is decidable.*

This theorem can alternatively be viewed from the perspective of ESO^2 on data trees where all nodes are annotated by distinct, linearly ordered data values. It then states that ESO^2 with access to the tree structure via MSO-definable atoms and with the ability to compare data values with respect to the linear order on data values is decidable over such trees. This should be compared with the decidability of EMSO^2 on data trees with possibly non-distinct data values, access to the tree structure via the children and sibling relation, as well as the ability to test whether two data values are equal [3].

An immediate consequence of Theorem 1.2 is the decidability of satisfiability of ESO^2 with two linear orders.

Corollary 1.3. *Satisfiability of* ESO^2 *with two linear orders is decidable.*

In Section 5 these results are stated more formally, and other consequences of Theorem 1.2 are discussed. We now briefly discuss our proof method.

Theorem 1.2 is proved by a reduction to the emptiness problem for our variant of register automata. To explain this reduction, let us consider the case of finite structures first. The finite satisfiability problem for two-variable logic

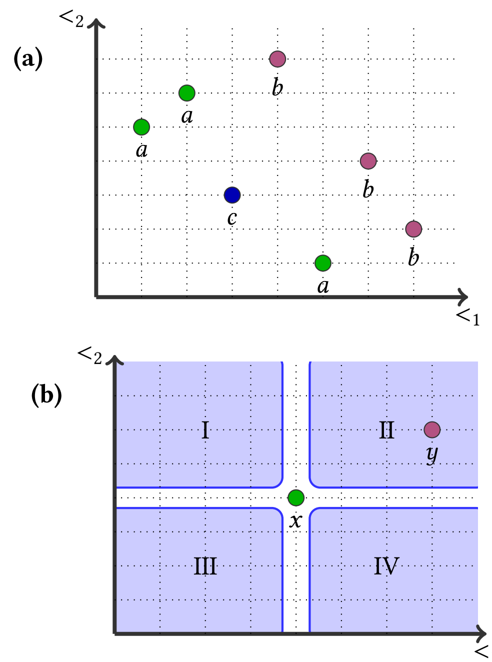

Figure 1. ***(a)*** *A structure with two linear orders represented as a point set in the two-dimensional plane. The structure satisfies the existential constraint $\forall x \exists y (a(x) \rightarrow (b(y) \wedge x <_1 y \wedge x <_2 y))$, and the universal constraint $\forall x \forall y \neg (a(x) \wedge b(y) \wedge y <_1 x \wedge y <_2 x)$. A linearly ordered data word corresponding to the structure is $(a,5)(a,6)(c,3)(b,7)(a,1)(b,4)(b,2)$.* ***(b)*** *In structures with two linear orders, constraints on an element x can be imposed in directions (I)–(IV). For instance, direction (II) corresponds to $x <_1 y \wedge x <_2 y$.*

with two linear orders, $\mathrm{FO}^2(<_1, <_2)$, can be reduced to an emptiness test for register automata on words in two steps: (1) exhibit a correspondence between structures with two linear orders and input data words for register automata, and (2) verify the conditions imposed by a $\mathrm{FO}^2(<_1, <_2)$-formula with a register automaton. For (1), finite structures with two linear orders $<_1$ and $<_2$ can be identified with finite data words with *distinct* data values from the rationals. Here, the unary types of single elements of the structure are represented as labels of the data word, the order $<_1$ corresponds to the linear order of positions in the word, and $<_2$ to the natural order of the rationals on data values, i.e., if positions x and y have data values p and q, respectively, then $x <_2 y$ if and only if $p < q$ (where $<$ is the natural linear order of the rationals). We refer to Figure 1 for an illustration of this correspondence.

Instead of directly verifying conditions imposed by an $\mathrm{FO}^2(<_1, <_2)$-formula, it is convenient to first convert such formulas into a set of *existential* and *universal constraints* [20]. An existential constraint enforces that for each element x of unary type σ there is an element y of unary type τ, such that y is in a specified direction from x with respect to $<_1$ and $<_2$, for instance in direction $x <_1 y$ and $x <_2 y$. A universal constraint can forbid patterns, that is, it can state that it is not the case that x and y are elements with unary types σ and τ, respectively, and y is in a specified direction of x (see Figure 1).

Such constraints can be easily verified by a register automaton. To this end, the automaton has registers $r^{\sigma}_{\max,\leftarrow}$, $r^{\sigma}_{\min,\leftarrow}$, $r^{\sigma}_{\max,\rightarrow}$, and $r^{\sigma}_{\min,\rightarrow}$, for each label σ, intended to store the maximal and minimal data value of σ-labeled positions to the left and right of the current position, respectively. The content of these registers can be guessed and verified by the automaton. Then, for determining whether an existential constraint such as the one above is satisfied, the automaton verifies that, for each σ-labeled position x, the register $r^{\tau}_{\max,\rightarrow}$ stores a value larger than the data value at x. Similarly universal constraints can be checked. Technically, the register automaton also has to ensure that all data values are distinct. While this is not possible in general, it can verify a weaker condition that guarantees that if some data word is accepted then so is one with distinct data values.

The above rough sketch can be used to obtain a new proof of decidability of finite satisfiability problem of FO^2 with two linear orders [20]. This automata-based approach generalizes well to various other results in this vein, by considering various domains $\mathbf{D}$ or various shapes of the input structures. The present paper is an illustration of the power of this approach.

To solve the general satisfiability of $FO^2(<_1, <_2)$ for linear orders $<_1$ and $<_2$, register automata need to be generalized in two directions. First, to allow infinite domains, we pass from finite data words to infinite objects with data, such as ω-words or infinite trees. Considering ω-words allows to study the case when $<_1$ is isomorphic to the naturals with their order, which does not cover all infinite orders. To encompass arbitrary countable linear orders, we move to infinite trees, as any countable order is isomorphic to a subset of the infinite complete binary tree with the left-to-right order on its nodes. More generally, any countable tree order can be encoded in the complete binary tree in a certain sense, so moving to infinite trees allows us to consider arbitrary countable tree orders $<_1$.

For the moment, let us focus on the case when $<_1$ is considered to be isomorphic to the naturals with their order, in which case we consider ω-words with data. Now, existential constraints coming from the FO^2 formula can enforce supremum-like conditions, e.g., they can require that (1) every σ-labeled element x has a τ-labeled element y with $x <_1 y$ and $x <_2 y$ and vice versa, that is, every τ-labeled element x has a σ-labeled element y with $x <_1 y$ and $x <_2 y$, and that (2) there is a ρ-labeled position that bounds from above all these τ and σ-labeled positions with respect to $<_2$. This is why we need to consider register automata over ω-words with data, which can access at each position the infimum and supremum of the values stored in a specified register in all future positions.

Finally, to solve the case of arbitrary countable orders $<_1$, we move from ω-words to infinite trees, as discussed above. Now, the infimum and supremum needs to be taken over all nodes of the tree which are descendants of the current node. This leads us to the study of tree register automata with suprema and infima constraints for infinite, binary trees with data. These are introduced in Section 3.

For the variants of the satisfiability problem briefly mentioned above – whether $<_1$ is a finite order, or isomorphic to the order of the naturals, or a countable order, or a tree order – the reduction of the satisfiability problem to the emptiness problem for the corresponding variant of automata always follows essentially the simple idea described above. This is described in Section 5.

On the other hand, deciding the emptiness problem for the corresponding models of register automata discussed above becomes more involved as the models are generalized. The overall idea of deciding emptiness tree register automata with extrema constraints is to (1) reduce to the emptiness problem for tree register automata without extrema constraints and subsequently (2) reduce the emptiness problem for such automata to the emptiness problem of parity automata over infinite trees. This is described in Section 4. Decidability then follows from the fact that the emptiness problem for parity automata is decidable [19].

2 Preliminaries

In this section we recapitulate basic notions and fix some of our notations.

A *tree* t is a prefix-closed subset of $\{0, 1\}^{\leq\omega}$, and each element $v \in t$ is called a node. The *ancestor order* of a tree t, denoted $<_{\text{anc}}$, is the strict prefix order on t.

We write $\swarrow v$ and $\searrow v$ for the left child $v0$ and the right child $v1$ of v; and denote the parent of v by $\uparrow v$. The subtree of t rooted at v is written as t_v. Assigning a label from a set Σ to every node of t yields a Σ*-labeled tree*.

A *(strict) partial order* $<$ over a domain $\mathbf{D}$ is a transitive, antisymmetric, and antireflexive relation, that is, if $a, b, c \in \mathbf{D}$ then $a < b$ and $b < c$ implies $a < c$; and $a < b$ and $b < a$ are never both satisfied at the same time. We use standard notions of *upper* and *lower bounds*, and of *suprema* and *infima* of subsets of a partially ordered set. A partial order is a *linear order* if $a < b$ or $b < a$ for all $a, b \in \mathbf{D}$ with $a \neq b$. A partial order $(\mathbf{D}, <)$ is a *tree order* if the set $\{y \mid y < x\}$ is totally ordered by $<$ for each $x \in \mathbf{D}$, and additionally, any two elements have some lower bound. For instance, all linear orders are tree orders, and the ancestor orders of trees are tree orders. In general, a tree order may not be isomorphic to the ancestor order of a tree. As examples consider a dense linear order, or an infinitely branching tree, or a combination of the two, where a dense linear order branches into infinitely many copies at each rational.

The *two-variable fragment of first-order logic* (short: FO^2) restricts first-order logic to the use of only two variables, which can be reused arbitrarily in a formula. The *two-variable fragment of existential second-order logic* (short: ESO^2) consists of all formulas of the form $\exists R_1 \ldots \exists R_k \varphi$ where $R_1, \ldots, R_k$ are relation variables and φ is a FO^2-formula. Since each FO^2-atom can contain at most two variables we assume, in the entire paper and without loss of generality, that all relation symbols are of arity at most two; see [6, page 5] for a justification. Restricting the set quantifiers of ESO^2 to be unary yields the fragment $EMSO^2$.

In this paper we study the satisfiability problems for ESO^2 and $EMSO^2$ with two orders $<_1$ and $<_2$, denoted by $ESO^2(<_1, <_2)$ and $EMSO^2(<_1, <_2)$, respectively. Often we will be interested in orders of a certain shape only, which we will then state explicitly. For instance, when saying "$ESO^2(<_1, <_2)$ for a tree order $<_1$ and a linear order $<_2$", we are interested in ESO^2 formulas with two distinguished binary relation symbols $<_1$ and $<_2$ which are always interpreted by a tree order and a linear order. In this paper, domains are always non-empty and, for convenience, we often identify relation symbols with their respective interpretations.

3 Tree Register Automata with Suprema and Infima Constraints

Register automata are finite state automata equipped with a set of registers that can store values from an infinite data domain. Here we introduce a variant of register automata for infinite trees and ordered domains. In the next section we will prove that their emptiness problem is decidable.

Our register automaton model is equipped with a mechanism for accessing infima and suprema of subtrees. Therefore it uses values from the domain $\overline{\mathbb{R}} \overset{df}{=} \mathbb{R} \cup \{-\infty, \infty\}$ linearly ordered in the natural way. The only feature of $\overline{\mathbb{R}}$ which will matter is that it is a dense linear order and contains all infima and suprema. Instead of $\overline{\mathbb{R}}$ we could equally well consider the real interval $[0, 1]$. We therefore fix the *ordered domain* $\mathbf{D} \overset{df}{=} \langle \overline{\mathbb{R}}, < \rangle$ which is a complete dense linear order with endpoints (treated as constants).

We consider a model of nondeterministic tree automata (with non-deterministic guessing of data values) which process infinite binary trees whose nodes are labeled by a label from a finite alphabet Σ and tuples of data values from $\mathbf{D}$, representing input values. Such an automaton has a finite set of registers storing values. It nondeterministically assigns to each node of the input tree a state from a finite state space and a valuation of the registers in the domain $\mathbf{D}$ (in particular, it allows for "guessing" of data values). At a node v, the transition relation has access to the automaton state as well as the Σ-label of v and its children $\swarrow v$, $\searrow v$, and it is capable of comparing the input numbers and the numbers stored in registers at those nodes using the linear order. Furthermore it can compare any of those register values to the infimum or supremum of data values in a given register at nodes in the subtree rooted at v reachable from v by a path whose labels satisfy a given regular property, e.g. of the form *supremum of values of register r at all descendants of v reachable by a path labeled by* a^*ba^*. The transition relation is described by a propositional formula, whose atomic formulas correspond to label and state tests, as well as register comparisons for the current node and its children, and the suprema and infima of register values in the current subtree.

More formally, a *tree register automaton with suprema and infima constraints* (short: *TRASI*) consists of the following components:

- a finite *input alphabet* Σ;
- a finite set of *input registers* I;
- a finite set of *states* Q;
- a finite set of *root states* $F \subseteq Q$;
- a finite set of *registers* R containing the input registers I;
- a function $\mathcal{L}$ mapping each register $r \in R$ to an associated regular language (specified by a nondeterministic finite state automaton);
- a nondeterministic *transition relation* δ which is given by a propositional formula with atomic formulas of the form
  - $\sigma, \swarrow\sigma, \searrow\sigma$ for $\sigma \in \Sigma$ and $q, \swarrow q, \searrow q$ for $q \in Q$, used for testing labels and states of the current node and its children nodes;
  - $s < t$ or $s = t$, used for comparing register values or suprema/infima: s and t range over the registers of the current node and the registers of its children nodes or are suprema or infima terms of the form $\sup r$ or $\inf r$, for $r \in R$ (that is, $s, t \in \{r, \swarrow r, \searrow r, \sup r, \inf r \mid r \in R\}$).
- a regular acceptance condition, given by a *parity function* $\Omega\colon Q \to \mathbb{N}$ assigning to each state its *rank*.

If a TRASI does not use suprema and infima terms then it is called a *tree register automaton.*

We next define inputs and runs for a tree register automaton $\mathcal{A}$ with suprema and infima constraints. An *input tree* for $\mathcal{A}$ is a complete infinite binary tree, whose vertices are labelled by elements from $\Sigma \times \mathbf{D}^I$. The labelling by $\mathbf{D}^I$ will form the register assignment for the input registers.

For a regular language $L \subseteq \Sigma^*$ an input tree t and two of its nodes v and w, we say that w is an *L-descendant* of v if w is a descendant of v (possibly v) and the labels along the path from v to w in t form a word which belongs to L. The terms $\sup r$ and $\inf r$, evaluated at a node v, will denote the supremum/infimum of all values of register r at nodes w which are L-descendants of v, where L is the language associated to register r. For technical reasons, we provide a piecemeal definition of a run of a TRASI, in which the values of the registers $\sup r$ and $\inf r$ are stored in auxiliary registers defined below, and then we require that the values of those registers are as expected.

Let $R_{\sup}$ and $R_{\inf}$ be two copies of the set R, where $R_{\sup} = \{r_{\sup} \mid r \in R\}$ and $R_{\inf} = \{r_{\inf} \mid r \in R\}$. Denote $\mathcal{R} = R \cup R_{\sup} \cup R_{\inf}$. Abusing language, elements of $R_{\sup}$ will be called *suprema registers*, and elements of $R_{\inf}$ will be called *infima registers.*

A *pre-run* ρ over an input tree t annotates each node of t by an element from $Q \times \mathbf{D}^{\mathcal{R}}$, so that registers from $I \subseteq \mathcal{R}$ get a value according to the input tree. The *state* at a node v in a pre-run ρ is the Q-component of $\rho(v)$. The labelling by $\mathbf{D}^{\mathcal{R}}$ will form the register assignment for the registers in $\mathcal{R}$. We write $\rho(v, r)$ for the value of register r in pre-run ρ; we often write $r(v)$ instead of $\rho(v, r)$ if ρ is clear from the context.

A pre-run ρ is *locally consistent* if each node v with left and right children $\swarrow v$ and $\searrow v$ satisfies δ with respect to ρ. Here, the satisfaction of the atomic formulas is defined as follows:

- σ, $\swarrow\sigma$, and $\searrow\sigma$ are satisfied in v if $v, \swarrow v$, and $\searrow v$ are labelled by $\sigma \in \Sigma$, respectively; similarly for q, $\swarrow q$ and $\searrow q$, for $q \in Q$;
- given a node v, a term of the form $r \in R$ evaluates to the value $\rho(v, r)$. A term of the form $\swarrow r$, for $r \in R$, evaluates to the value $\rho(\swarrow v, r)$, and likewise for $\searrow r$. Finally, a term of the form $\sup r$ evaluates to $\rho(v, r_{\sup})$, and a term of the form $\inf r$ evaluates to $\rho(v, r_{\inf})$.
- if s, t are two terms as above, then the atomic formula $s < t$ is satisfied in the node v if the value of the term s is smaller than the value of the term t. The definition for $s = t$ is analogous.

Satisfaction for boolean operations is defined as usual.

A *run* is a locally consistent pre-run which additionally satisfies the following consistency requirement: for every node v, $\rho(v, r_{\sup})$ is the supremum of the values $\rho(w, r)$, where w ranges over all L-descendants of v where L is the language associated to r, whereas $\rho(v, r_{\inf})$ is the infimum of all such values. If no L-descendants exist, then the supremum is $-\infty$ and infimum is $+\infty$ by definition. A pre-run ρ is *accepting* if the state at the root belongs to F, and every branch of ρ satisfies the parity condition, i.e., on every (infinite, rooted) branch of ρ, if the states at the nodes along this branch are $q_1, q_2, \ldots$, then $\limsup_{n\to\infty}(\Omega(q_n))$ is even.

A TRASI *accepts* an input tree t if it has an accepting run over that tree. The *emptiness problem* for TRASI is the problem of deciding whether a given TRASI accepts some input tree. In the next section we prove that this problem is decidable.

Note that instead of equipping the automaton model with a parity acceptance condition, we could equip it with an MSO acceptance condition, i.e. an MSO formula ϕ using two binary predicates: s_1 and s_2, standing for left and right successor (in the binary tree), and for each $q \in Q$ a unary predicate λ_q, holding at nodes labeled with q. This would not change the expressive power of the tree automata, since parity tree automata have the same expressive power as MSO over infinite binary trees [19, 23].

Example 3.1. We construct an automaton that recognizes trees with one data value per node such that the values on each infinite path form a strictly increasing sequence of numbers. The data value is stored in an input register r. The automaton has two states $q_\top$ and $q_\bot$, where $q_\top$ represents the situation where the sequence of the numbers from the root to the current node is increasing; $q_\top$ is the only root state. A path is accepted if $q_\top$ occurs infinitely often, i.e. the parity function Ω is defined as $\Omega(q_\top) = 2$ and $\Omega(q_\bot) = 1$.

The transition relation is defined by

$$\left(\swarrow q_\top \leftrightarrow (q_\top \wedge r < \swarrow r)\right) \wedge \left(\searrow q_\top \leftrightarrow (q_\top \wedge r < \searrow r)\right)$$

where the first part checks that the state is propagated properly to the left child, and the second part does the same for the right child.

4 Deciding Emptiness

In this section we prove the following:

Theorem 4.1. *The emptiness problem for tree register automata with suprema and infima constraints is decidable.*

Given a TRASI $\mathcal{A}$, we construct (in Section 4.1) a tree register automaton $\widehat{\mathcal{A}}$ (without extrema constraints) such that $\mathcal{A}$ has an accepting run if and only if $\widehat{\mathcal{A}}$ has an accepting run. As emptiness of register automata over infinite data trees is decidable (see Section 4.2), this will yield decidability of the emptiness problem for TRASI.

4.1 From TRASI to Tree Register Automata

For the sake of readability and due to space constraints, we prove how $\widehat{\mathcal{A}}$ can be constructed when all languages associated with registers in the TRASI $\mathcal{A}$ are equal to Σ^*, that is, if suprema and infima of data values are taken over all nodes of a subtree. The proof for general regular languages is deferred to the full version of the paper.

We would like to construct $\widehat{\mathcal{A}}$ so that it accepts exactly those pre-runs of $\mathcal{A}$ which are actual runs. This requirement, however, is too strong. The reason is that applying an arbitrary monotone bijection of $\mathbf{D}$ to an accepting run of a register automaton always yields an accepting run, whereas the same property fails for accepting runs of TRASI's, since such a bijection might not preserve suprema and infima.

For this reason $\widehat{\mathcal{A}}$ checks a weaker condition. If the supremum of a register r at a node v is c, then (if c is not present in t_v) there must be an infinite path starting from v such that the supremum of r has value c for all nodes along this path. Then one can find a sequence of nodes $v_1, v_2, \ldots$ on the path that have witness nodes $w_1, w_2, \ldots$ below them (but not necessarily on the path) whose r-values tend to c. The intention of $\widehat{\mathcal{A}}$ is to check for the existence of these witnesses $w_1, w_2, \ldots$. As the automaton cannot check that their data values actually tend to the supremum c, it chooses the witnesses $w_1, w_2, \ldots$ so that at least they exceed all those values currently stored in the registers which are smaller than c. This is formalized below.

First, we impose a normalization assumption on the TRASI $\mathcal{A}$. Namely, say that a TRASI $\mathcal{A}$ is *normalized* if the transition relation does not allow directly comparing values in the left and right child of a given node v, i.e., it does not contain atoms of the form $\searrow r < \swarrow s$ or $\swarrow r < \searrow s$, for $r, s \in R$. It is easy to see that any TRASI can be converted into a normalized TRASI which accepts the same input trees (possibly at the cost of introducing new registers).

Let ρ be a pre-run of $\mathcal{A}$. Say that ρ is *extrema-consistent* if for every node v and $r \in R$,

$$r_{\sup}(v) = \max(r(v), r_{\sup}(\swarrow v), r_{\sup}(\searrow v)),$$
$$r_{\inf}(v) = \min(r(v), r_{\inf}(\swarrow v), r_{\inf}(\searrow v)).$$

Clearly, every run is extrema-consistent, and a tree register automaton (without extrema constraints) can easily verify that a given pre-run is extrema-consistent and locally consistent. Because of this,

> *from now on, we assume that every considered TRASI is normalized, and that all pre-runs are locally consistent and extrema-consistent.*

The existence of a pre-run as above does not yet guarantee the existence of a run, since the extrema might not be approached by actual values of registers. For example, it might be the case that $r_{\sup}(v) = 5$ for all nodes v, whereas $r(v) = 4$ for all nodes v. This case can be easily detected by a register automaton, which checks that whenever $r(v) < r_{\sup}(v)$, then there is some descendant v' of v with $r(v) < r(v') < r_{\sup}(v)$. Now suppose that $r_{\sup}(v) = 5$ and $r(v) = 4 - 1/d$ for all nodes at depth d, which is still not an actual run. Insofar as the register r is concerned, this pre-run can be modified into a run by simply replacing $r_{\sup}(v)$ by 4. However, suppose there is another register s such that $s(w) = 4.5$ for some w and that the inequalities $r(w) < s(w) < r_{\sup}(w)$ are enforced by the acceptance condition. Then we cannot easily fix this pre-run to obtain a run, since $s(w)$ separates the values $r(v)$ from the value $r_{\sup}(v)$. To make sure that such a situation does not occur, we introduce witness families which ascertain that the values $r(v)$ exceed all values $s(w)$ which are encountered along a branch. This is formalized as follows.

For each register $r \in R$ introduce two symbols r_+ and r_-. Let $R^\pm = \{r_+, r_- \mid r \in R\}$ be the collection of all such symbols, and fix an arbitrary total order on $R^\pm$.

Let $\mathcal{P}$ be a partition of the set of nodes of the tree underlying ρ into finite paths. Say that a path ρ in $\mathcal{P}$ is the *father* of a path ρ' if the father of the smallest node (wrt. the ancestor order) in ρ' belongs to ρ. To each path in ρ assign a label in $R^\pm$ in a cyclic fashion, as follows. Say that a labeling $\lambda\colon \mathcal{P} \to R^\pm$ is *cyclic* if:

- the label $\lambda(\rho)$ of the path ρ containing the root is the smallest label in $R^\pm$, and
- if a path ρ is the father of ρ' then the label $\lambda(\rho')$ is the successor of the label $\lambda(\rho)$, in cyclic order, according to the fixed order on $R^\pm$.

Clearly, there is exactly one cyclic labeling of $\mathcal{P}$. Henceforth, when considering a partition $\mathcal{P}$ as above, we assume each path in $\mathcal{P}$ is labeled according to the cyclic labeling.

A path $\rho \in \mathcal{P}$ is called a sup-*witness path* associated to r if $\lambda(\rho) = r_+$ and it is called a inf-*witness path* associated to r if $\lambda(\rho) = r_-$. For a witness path π associated to r, its *target value* is $r(w)$, where w is the largest node in π wrt. the ancestor order. A witness path *starts* at its smallest node wrt. the ancestor order. If π is a witness path associated to r which starts at v, then the *associated extremum* is $r_{\sup}(v)$ if π is a sup-witness path, and $r_{\inf}(v)$ if π is an inf-witness path.

Note that if $\lambda(\rho) = r_+$ then the target value of ρ is at most equal to the associated extremum of ρ, since ρ is extrema-consistent (if $\lambda(\rho) = r_-$, then the inequality is reversed). The *gap* of a witness path with target value c and associated extremum d is the interval with endpoints c (inclusively) and d (exclusively). It is empty precisely when the target value attains the extremum.

Fix a partition $\mathcal{P}$ of the set of nodes into finite paths. We now define when $\mathcal{P}$ is a certificate. Intuitively, we require that the target values of the witness paths are sufficiently close to the associated extrema, i.e., each gap is sufficiently small. Formally, the partition $\mathcal{P}$ is a *certificate* if each witness path π satisfies the following *gap condition*, where v denotes the starting node of π:

> *the gap of π does not contain any value $c \in \mathbf{D}$ such that $c = s(v)$ or $c = s(\uparrow v)$ for some $s \in \mathcal{R}$.*

Below we prove the following three properties of certificates:

1. every run has a certificate,
2. if some accepting pre-run of $\mathcal{A}$ has a certificate then $\mathcal{A}$ has some accepting run,
3. a tree register automaton can verify if a given pre-run has a certificate.

These properties together easily yield Theorem 4.1.

To show that every run of $\mathcal{A}$ has a certificate, we greedily add paths to the family $\mathcal{P}$ without violating the gap condition, as shown below.

Lemma 4.2. *Every run of $\mathcal{A}$ has a certificate.*

Proof. Fix a run ρ. We construct the family $\mathcal{P}$ in stages. Initially, $\mathcal{P}$ is empty, and each stage proceeds as follows. Pick a minimal node v such that $v \notin \bigcup \mathcal{P}$. If the node v is the root then let $l \in R^\pm$ be the smallest label in $R^\pm$. Otherwise, the parent of v already belongs to some path $\rho \in \mathcal{P}$ which has an assigned label l', and let $l \in R^\pm$ be the successor of l' in cyclic order, according to the fixed order on $R^{\pm}$. Assume $l = r_+$ for some $r \in R$. The case when $l = r_-$ is treated symmetrically. Pick any descendant w of v such that the interval with endpoint $r(w)$ (inclusively) and $r_{\sup}(v)$ (exclusively) does not contain any value $c \in \mathbf{D}$ such that $c = s(v)$ or $c = s(\uparrow v)$ for some $s \in \mathcal{R}$. Add the path joining v with w to the family $\mathcal{P}$ and associate the label l with it. Proceed to the next stage.

By construction, in the limit we obtain a certificate. □

The key point of certificates is that the existence of a certificate implies the existence of a run.

Lemma 4.3. *For every (locally consistent, extrema-consistent) pre-run ρ which has a certificate there is a run ρ' whose states agree with the states of ρ.*

Proof. To prove the lemma, fix a pre-run ρ and some certificate $\mathcal{P}$. We construct a run ρ' which has the same states as ρ. The run ρ' is obtained by successively processing all nodes v, starting from the root, and shifting the register values in their subtrees, without changing local relationships. Towards this goal, let $v_0, v_1, \ldots$ be an enumeration of the tree nodes in which every node appears after all its ancestors (in particular, v_0 is the root). We construct a sequence $\rho_{v_0}, \rho_{v_1}, \ldots$ of pre-runs such that each run has the same states as ρ, and such that the sequence converges to a run ρ'. Here, convergence means that for every node v, the sequence $\rho_{v_0}(v), \rho_{v_1}(v), \ldots$ of labels in $Q \times \mathcal{R}$ assigned to v is ultimately equal to $\rho'(v)$.

Define $\rho_{v_0} \overset{df}{=} \rho$. We describe, for $v \overset{df}{=} v_i$ and $u \overset{df}{=} v_{i-1}$, how ρ_v is constructed, assuming that ρ_u has already been constructed. The register values of the subtree t_v will be shifted, and the assignment will not change on nodes from $t \setminus t_v$. This guarantees that the sequence $\rho_{v_0}, \rho_{v_1}, \ldots$ indeed converges to some pre-run ρ'.

The pre-run ρ_v is defined as follows. Outside of the subtree t_v, it agrees with the pre-run ρ_u. Inside the subtree t_v, an arbitrary monotone bijection $f : \mathbf{D} \to \mathbf{D}$ with the following properties is applied to all register values of nodes in t_v in the pre-run ρ_u:

1. $f(c) = c$ for all $c \in \mathbf{D}$ such that $c = s(v)$ or $c = s(\uparrow v)$, for some $s \in \mathcal{R}$, and
2. if $c \in \mathbf{D}$ is the target value of a witness path π starting at v with associated extremum d, then $|f(c) - f(d)| < \frac{1}{n}$, where n is the depth of the node v (note that $c = f(c)$ by the first item).

Intuitively, the bijection f shrinks the gap of π so that its length is smaller than $\frac{1}{n}$. As the gap of π did not contain any values of the form $s(v)$ or $s(\uparrow v)$ by definition of a certificate, a monotone bijection f with the above properties exists. This ends the description of ρ_v.

The above construction preserves the following invariant:

- ρ_v is a (locally consistent, extrema-consistent) pre-run with the same states as ρ,
- ρ_v has certificate $\mathcal{P}$.

It follows that the sequence of pre-runs $\rho_{v_0}, \rho_{v_1}, \ldots$ converges to a pre-run ρ' with the above properties. It remains to show that ρ' is a run, i.e., that for every node v and register r,

$$r_{\sup}(v) = \sup\{r(v') \mid v' \textit{ is a descendant of } v\} \tag{1}$$

$$r_{\inf}(v) = \inf\{r(v') \mid v' \textit{ is a descendant of } v\}. \tag{2}$$

We prove (1) while (2) is proved analogously. The inequality $\geq$ in (1) follows from the fact that ρ' is extrema-consistent.

Towards proving $\leq$, pick an inclusion-maximal branch π starting at v, such that $r_{\sup}(v) = r_{\sup}(v')$ for all $v' \in \pi$. If the branch π is finite then $r_{\sup}(v) = r(w)$ where w is the largest node in π (wrt. the ancestor order) and $r_{\sup}(\swarrow w), r_{\sup}(\searrow w)$ are both smaller than $r(w)$. By extrema-consistency of ρ' it follows that the right-hand side in (1) is equal to $r(w)$, which proves (1) as $r_{\sup}(v) = r(w)$.

Suppose now that π is infinite. To prove (1), it suffices to exhibit a sequence $w_1, w_2, \ldots$ of descendants of v such that

$$r_{\sup}(v) = \lim_{n \to +\infty} r(w_n). \tag{3}$$

Construct sup-witness paths $\pi_0, \pi_1, \ldots$ associated to r, as follows. Assuming we have constructed $\pi_0, \pi_1, \ldots, \pi_{n-1}$, the path π_n is any sup-witness path associated to r such that its starting node u_n is in π, is larger than u_{n-1}, and does not belong to $\pi_0 \cup \ldots \cup \pi_{n-1}$. It is easy to see that such a node u_n exists, by the fact that $\mathcal{P}$ is a partition into finite paths, and the fact that their labeling is cyclic.

Then the nodes $v = u_0, u_1, \ldots$ all lie on the path π. In particular, $r_{\sup}(v) = r_{\sup}(u_n)$ and u_n has depth at least n, for all n. Fix n and let w_n be the largest node in π_n wrt. the ancestor order. By the second item in the definition of the bijection f obtained when defining the pre-run ρ_{u_n}, the following holds for that pre-run:

$$|r(w_n) - r_{\sup}(v)| = |r(w_n) - r_{\sup}(u_n)| < \frac{1}{n}. \tag{4}$$

We observe that the same inequality holds for the limit pre-run ρ'. This follows from the first item in the definition of the bijection f obtained when defining the pre-runs ρ_w for all w which are descendants of u_n.

Since the inequality (4) holds for the pre-run ρ' and for all n, this proves (3), yielding (1). Hence ρ' is a run. □

It remains to show that the existence of certificates can be decided by tree register automata. This is proved by encoding the data of a certificate for a pre-run ρ using a finite labeling (for marking starting points and labels of witness paths) and one register (storing at a node v the target value of the witness path containing v) and verifying that they form a valid certificate: a parity condition for checking that the paths are finite, and inequality constraints on the target values of the witness paths to verify that they satisfy the gap condition of a certificate.

Lemma 4.4. *There is a register automaton $\mathcal{B}$ which accepts a pre-run ρ of $\mathcal{A}$ if and only if it has a certificate. Moreover, $\mathcal{B}$ can be constructed in polynomial time, given $\mathcal{R} = R \cup R_{\sup} \cup R_{\inf}$.*

Proof sketch. The automaton $\mathcal{B}$ guesses a certificate for ρ, as follows. A partition $\mathcal{P}$ of all vertices into connected sets can be represented as the set $X = \{\min P \mid P \in \mathcal{P}\}$ of least elements of the sets in $\mathcal{P}$. Then $\mathcal{P}$ is a partition into finite paths if and only if every branch contains infinitely many elements of X and every node has at least one child in X.

The automaton $\mathcal{B}$ guesses the partition $\mathcal{P}$, represented by the set X as described above, and verifies that it is a partition into finite paths. Additionally, each node is nondeterministically labeled by a label in $R^{\pm}$ and the automaton verifies that nodes in the same part of $\mathcal{P}$ have equal labels, and that this yields a cyclic labeling of $\mathcal{P}$. The automaton has a single register t. At each node, it nondeterministically selects a value for this register and verifies that the value $t(v)$ is the same for all nodes v in the same part of $\mathcal{P}$, and that $t(v) = r(v)$ if v is the largest node in its part (wrt. the ancestor order) and is labeled r_+ or r_-. Hence, $t(v)$ represents the target value of the witness path associated to r and containing v.

It remains to verify that the guessed partitions form a certificate for ρ. To this end, the automaton verifies, for each node v and register r, that the interval with endpoint $t(v)$ (inclusively) and $r_{\sup}(v)$ (exclusively) does not contain any value $c \in \mathbf{D}$ such that $c = s(v)$ or $c = s(\uparrow v)$ for some $s \in \mathcal{R}$.

If all the above is confirmed, the automaton accepts. By construction, $\mathcal{B}$ accepts ρ if and only if ρ has a certificate. □

The special case of Theorem 4.1 where all associated languages are Σ^* now follows easily, as we now describe. Due to space constraints, the general proof is deferred to the full version of the paper.

We give a reduction from the emptiness problem for TRASI to the emptiness problem for tree register automata. That the latter problem is decidable is shown in the next section.

Fix a TRASI $\mathcal{A}$. We construct a tree register automaton $\widehat{\mathcal{A}}$ such that $\mathcal{A}$ has an accepting run if and only if $\widehat{\mathcal{A}}$ has an accepting run.

First, construct the tree register automaton $\mathcal{B}$ as given by Lemma 4.4. Then, construct an automaton $\widehat{\mathcal{A}}$ which is the composition of $\mathcal{A}$ and $\mathcal{B}$: given a labeled input tree t, the automaton $\widehat{\mathcal{A}}$ nondeterministically selects a pre-run ρ of $\mathcal{A}$, tests that it is locally consistent, extremum-consistent, satisfies the parity condition, and uses $\mathcal{B}$ to verify that ρ has a certificate. If all those tests are passed, then $\widehat{\mathcal{A}}$ accepts. Note that $\widehat{\mathcal{A}}$ is a tree register automaton (without extrema constraints).

Then $\widehat{\mathcal{A}}$ has an accepting run if and only if $\mathcal{A}$ has an accepting run. In one direction, suppose $\mathcal{A}$ has an accepting run ρ on some input tree t. Then $\widehat{\mathcal{A}}$ also accepts t, as witnessed by the pre-run ρ, which is accepted by $\mathcal{B}$ by Lemma 4.2. Conversely, suppose that $\widehat{\mathcal{A}}$ has an accepting run. This means that there is a pre-run ρ of $\mathcal{A}$ which is extrema-consistent, satisfies the parity condition, and has a certificate. By Lemma 4.3, there is a run ρ' of $\mathcal{A}$ whose states agree with the states of ρ. In particular, ρ' satisfies the parity condition, so is an accepting run of $\mathcal{A}$.

This completes the reduction, and proves Theorem 4.1 in the special case where the languages associated to the registers are trivial.

4.2 Emptiness for Tree Register Automata is Decidable

The last step in the proof of Theorem 4.1 is an emptiness test for tree register automata. The proof is along the same lines as the proof for register automata on finite words. The result is folklore, but we provide the proof for the sake of completeness.

Lemma 4.5. *Emptiness of tree register automata is decidable.*

We will show that one can effectively construct, from each tree register automaton $\mathcal{A}$, a parity tree automaton $\mathcal{A}'$ such that $\mathcal{A}$ is non-empty if and only if $\mathcal{A}'$ is non-empty. One obvious problem is that $\mathcal{A}$ is over an infinite alphabet, while $\mathcal{A}'$ needs to work over a finite alphabet. This problem can be tackled by a standard technique: the automaton $\mathcal{A}'$ abstracts the alphabet and the data values of an input tree for $\mathcal{A}$ into *types* that capture the essential information.

For preciseness, let $\mathcal{A}$ be a tree register automaton with input alphabet Σ and input registers I. As we are interested in emptiness, we may assume that I contains all registers of $\mathcal{A}$. Then each node of an input tree t for $\mathcal{A}$ is labelled by an element from $\Sigma \times \mathbf{D}^I$.

Consider a node v of t with child nodes $\swarrow v$ and $\searrow v$. Then the *type* $\textsc{tp}(v)$ of v is the set of all satisfied propositional formulas with variables of the form σ, $\swarrow\sigma$, and $\searrow\sigma$, for all $\sigma \in \Sigma$ as well as $s < t$ where $s, t \in \{r, \swarrow r, \searrow r \mid r \in I\}$. Here satisfaction is defined as before. The set of possible types is finite and henceforth denoted Σ_{Types}.

The input tree t can be converted into a *type tree* $\textsc{tp}(t)$ over Σ_{Types} as follows. The tree $\textsc{tp}(t)$ has the same set of nodes as t and each node v is labelled by $\textsc{tp}(v)$.

Lemma 4.6. *One can construct a parity tree automaton $\mathcal{A}'$ over Σ_{Types} such that for any tree t over $\Sigma \times \mathbf{D}^I$,*

$$\mathcal{A} \text{ accepts } t \quad \text{if and only if} \quad \mathcal{A}' \text{ accepts } \textsc{tp}(t).$$

Lemma 4.7. *One can construct a parity automaton $\mathcal{B}'$ over Σ_{Types} such that for every tree t' over Σ_{Types}:*

$$\mathcal{B}' \text{ accepts } t' \text{ if and only if } t' = \textsc{tp}(t) \text{ for some tree } t \text{ over } \Sigma \times \mathbf{D}^I.$$

Proof (of Lemma 4.5). From the above two lemmas it follows that $\mathcal{A}$ is non-empty if and only if there is a tree t' over Σ_{Types} which is accepted both by $\mathcal{A}'$ and $\mathcal{B}'$. Indeed, if t' is accepted by $\mathcal{A}'$ and $\mathcal{B}'$, then there is a tree t over $\Sigma \times \mathbf{D}^I$ with $t = \textsc{tp}(t')$ by Lemma 4.7 which is accepted by $\mathcal{A}$ due to Lemma 4.6. The other direction is immediate.

Since parity automata are closed under intersection, there is a parity tree automaton C' which accepts a tree t' if and only if it is accepted both by $\mathcal{A}'$ and by $\mathcal{B}'$. Therefore, deciding emptiness of $\mathcal{A}$ is equivalent to deciding emptiness of C'. □

5 Satisfiability of Two-Variable Logic with two Orders

In this section we present a decision procedure for two variable logic over countable structures that include one tree order $<_1$, one linear order $<_2$, and access to binary atoms definable in MSO over $<_1$. Our decision procedure uses the emptiness test for tree register automata with suprema and infima constraints developed in the preceding section.

Let $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$ denote the set of $\mathrm{EMSO}^2(<_1, <_2)$ formulas that can use binary atoms definable in $\mathrm{MSO}(<_1)$ in addition to atoms from $\{<_1, <_2\}$. Below, we consider the problem of deciding whether a given formula of this logic is satisfied in some countable structure equipped with a tree order $<_1$ and linear order $<_2$.

Theorem 5.1. *For the logic* $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$*, where* $<_1$ *is a tree order and* $<_2$ *is a linear order, the countable satisfiability problem is decidable.*

The theorem implies a decision procedure for general satisfiability (i.e., unrestricted to countable domains) for the logic $\mathrm{EMSO}^2(\mathrm{FO}(<_1), <_2)$, i.e., if the additional atoms are restricted to be definable in FO. This follows from the observation that the satisfiability of a formula $\exists \bar{Z}\varphi$, where φ is first-order, is equivalent to (general) satisfiability of φ (as a formula over extended signature $\{<_1, <_2, \bar{Z}\}$); and the downward Löwenheim-Skolem theorem for first-order logic.

Corollary 5.2. *Satisfiability of* $\mathrm{EMSO}^2(\mathrm{FO}(<_1), <_2)$ *for a tree order* $<_1$ *and a linear order* $<_2$ *is decidable.*

Furthermore, the theorem implies decision procedures for EMSO^2 with MSO-definable and FO-definable classes of orders $<_1$. We give two example corollaries.

Corollary 5.3. *Satisfiability of* $\mathrm{EMSO}^2(<_1, <_2)$ *is decidable when* $<_2$ *is a linear order and* $<_1$ *is (i) a linear order, or (ii) the natural linear order on the naturals, or (iii) the natural linear order on the integers.*

Proof. For a tree order $<_1$ one can axiomatize in FO that it is a linear order. Applying Corollary 5.2 yields statement (i). The natural order on the naturals and integers can be axiomatised in MSO, and thus statements (ii) and (iii) follow from Theorem 5.1. □

Using techniques developed by Pratt-Hartmann in [18] for partial orders, one can reduce satisfiability of $\mathrm{ESO}^2(<_1, <_2)$ to satisfiability of $\mathrm{EMSO}^2(<_1, <_2)$ when $<_1$ and $<_2$ are linear orders. Combining this reduction with Corollary 5.2 yields the following result.

Corollary 5.4. *General satisfiability of* $\mathrm{ESO}^2(<_1, <_2)$ *for linear orders* $<_1$ *and* $<_2$ *is decidable.*

The proof of Theorem 5.1 is provided in Section 5.1. The proof strategy for Corollary 5.4 is presented in Section 5.2.

5.1 Decision Procedure for a Tree Order and a Linear Order

The main ingredient for the proof is a similar decidability result, for the the special case when $<_1$ is restricted to be the ancestor order of the complete binary tree. Combining this with the fact that every countable tree order can be interpreted in the ancestor order of the complete binary tree yields Theorem 5.1.

Proposition 5.5. *For the logic* $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$*, where* $<_1$ *is the ancestor order of the full binary tree and* $<_2$ *is a linear order, the countable satisfiability problem is decidable.*

Proof sketch. The high level idea for testing countable satisfiability of an $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$ sentence φ is to construct, from φ, a TRASI $\mathcal{A}$ such that φ has a model if and only if $\mathcal{A}$ accepts some input tree. The automaton $\mathcal{A}$ will work on data trees with a single data value per node. Intuitively the tree order of such an input tree will encode the order $<_1$ in φ and the order on data values will encode the order $<_2$. In this intuition we assume that all data values of data trees are distinct: while the automaton cannot ensure this, we will later see a workaround.

The automaton $\mathcal{A}$ is not constructed from φ directly but from an equisatisfiable formula ψ of simpler shape obtained as follows. The formula φ can be transformed into an equisatisfiable formula

$$\varphi' = \exists \bar{Z}\Big(\forall x \forall y \psi(x, y) \wedge \bigwedge_i \forall x \exists y \psi_i(x, y)\Big)$$

in Scott normal form (see, e.g., [21] and [7, page 17]) where ψ and all ψ_i are quantifier-free formulas whose atoms are $\mathrm{MSO}(<_1)$-formulas and relation symbols from $\bar{Z} \cup \{<_2\}$.

Under the assumption that $<_2$ is a linear order, the formula φ' can be further normalized. Using standard techniques, it can be transformed into an equisatisfiable formula of the form $\varphi'' = \exists \bar{Z}\psi$ where ψ is a conjunction of *existential constraints* of the form

$$\forall x \exists y\Big(\bigvee_i \big(x <_2 y \wedge \eta_i(x, y)\big) \vee \bigvee_i \big(x >_2 y \wedge \vartheta_i(x, y)\big)\Big)$$

and of *universal constraints* of the form

$$\forall x \forall y\big(x <_2 y \rightarrow \eta(x, y)\big)$$

where η_i, ϑ_i, and η are arbitrary $\mathrm{MSO}(<_1)$ formulas with free variables x and y which may use relation symbols from $\bar{Z}$. Note that the constraints would involve other $\{<_2\}$-types if $<_2$ was not a linear order. The formulas φ'' and ψ are equisatisfiable (where the signature of ψ extends the signature of φ'' by the relation symbols in $\bar{Z}$).

$$\alpha(v, r_{\tau,\sup}) = \max\left\{\begin{array}{rl} & \{d(v) \mid \text{composing } \uparrow\delta(v), \swarrow\delta(v), \searrow\delta(v), \text{and the type of } v \text{ yields } \tau\} \\ \cup & \{\alpha(v, r_{\gamma,\sup,\uparrow}) \mid \text{composing } \gamma, \swarrow\delta(v), \searrow\delta(v), \text{and the type of } v \text{ yields } \tau\} \\ \cup & \{\alpha(v, r_{\gamma,\sup,\swarrow}) \mid \text{composing } \uparrow\delta(v), \gamma, \searrow\delta(v), \text{and the type of } v \text{ yields } \tau\} \\ \cup & \{\alpha(v, r_{\gamma,\sup,\searrow}) \mid \text{composing } \uparrow\delta(v), \swarrow\delta(v), \gamma, \text{and the type of } v \text{ yields } \tau\} \end{array}\right\}$$

$$\alpha(v, r_{\gamma,\sup,\uparrow}) = \max\left\{\begin{array}{rl} & \{d(w) \mid \text{composing } \uparrow\delta(w), \searrow\delta(w), \text{and the type of } w \text{ yields } \gamma\} \\ \cup & \{\alpha(w, r_{\gamma',\sup,\uparrow}) \mid \text{composing } \gamma', \searrow\delta(w), \text{and the type of } w \text{ yields } \gamma\} \\ \cup & \{\alpha(w, r_{\gamma',\sup,\searrow}) \mid \text{composing } \uparrow\delta(w), \gamma', \text{and the type of } w \text{ yields } \gamma\} \end{array}\right\}$$

Figure 2. *Equations for verifying $\alpha(v, r_{\tau,\sup})$ and $\alpha(v, r_{\gamma,\sup,\uparrow})$ in the proof of Proposition 5.5. Here $w \stackrel{df}{=} \uparrow u$.*

Let Θ be the signature of ψ and Σ the set of unary types over Θ. From now on in this proof, all data trees are over Σ with a single data value per node.

The automaton $\mathcal{A}$ will be constructed such that it accepts a data tree over Σ if and only if ψ is satisfiable. Even more, there will be a one-to-one correspondence between data trees t with distinct data values accepted by $\mathcal{A}$ and models of ψ. We next outline this correspondence and then construct $\mathcal{A}$.

Each structure $\mathcal{S}$ over Θ corresponds to a data tree $\textsc{tree}(\mathcal{S})$ (over Σ, only one data value per node) with distinct data values as follows: the nodes of $\textsc{tree}(\mathcal{S})$ are the elements of the universe of $\mathcal{S}$ labeled by their unary type in $\mathcal{S}$; the tree order of $\textsc{tree}(\mathcal{S})$ is given by $<_1$; and the order of the data values is given by $<_2$, that is, data values are assigned such that $d(x) < d(y)$ if and only if $x <_2 y$ for all nodes x and y. Since $<_2$ is a linear order, all nodes of $\textsc{tree}(\mathcal{S})$ have distinct data values.

On the other hand, a data tree t with distinct data values corresponds to the structure $\textsc{structure}(t)$ whose universe contains all nodes of t; unary relations are chosen according to the Σ-labels of t; and $<_1$ and $<_2$ are interpreted according to the tree order of t and the order of the unique data values, respectively.

For a data tree t with distinct data values, we say that t satisfies ψ if its corresponding structure $\textsc{structure}(t)$ satisfies ψ. The automaton $\mathcal{A}$ is constructed such that, for data trees t with distinct values, it accepts t if and only if t satisfies ψ. Furthermore, if some data tree is accepted, then so is a data tree with distinct data values. This ensures that a data tree is accepted if and only if ψ has a model.

The automaton $\mathcal{A}$ is the intersection of two automata $\mathcal{A}_\psi$ and $\mathcal{A}_{\neq}$. The automaton $\mathcal{A}_\psi$ verifies that an input tree satisfies ψ under the assumption that all data values are distinct, but with undefined behaviour for input trees with distinct nodes with the same data value. The automaton $\mathcal{A}_{\neq}$ ensures that if a tree is accepted by $\mathcal{A}_\psi$ (with possibly the same data value on more than one node) then so is a tree with distinct data values on all nodes. Note that TRASIs can be easily seen to be closed under intersection.

In the rest of this proof, for a node v of an input tree, we denote the data value of v by $d(v)$.

We first explain the construction of the automaton $\mathcal{A}_\psi$. This automaton checks whether an input tree satisfies each of the existential and universal constraints in ψ. To this end, it will (1) guess and verify MSO-type information for each node, in order to handle the MSO-formulas η_i, ϑ_i and η in the constraints, and (2) use additional registers for checking the constraints.

Recall that a k-ary MSO-type $\tau(x_1, \ldots, x_k)$ of quantifier-rank at most q is a maximally consistent subset of MSO-formulas of quantifier-rank q and free variables $x_1, \ldots, x_k$ (see [13, Section 7.2] for background on MSO-types). We write $(u_1, \ldots, u_k) \models \tau$ if the tuple $(u_1, \ldots, u_k)$ of a data tree is of type τ. It is folklore that the rank-d MSO-type of a tree can be constructed from the rank-d MSO-types of its components. This is implicit in Shelah's Theorem on *generalised sums* [22], see also the exposition from Blumensath et al. [2].

The automaton $\mathcal{A}_\psi$ guesses and verifies the following type information. Suppose that $q \in \mathbb{N}$ is the maximal quantifier-rank of the formulas η_i, ϑ_i and η. For each node v the automaton guesses the following unary MSO-types of quantifier-rank q:

- the type $\uparrow\delta(v)$ of $\uparrow v$ in $t \setminus t_v$,
- the type $\swarrow\delta(v)$ of $\swarrow v$ in $t_{\swarrow v}$, and
- the type $\searrow\delta(v)$ of $\searrow v$ in $t_{\searrow v}$.

Those guesses can be verified since tree register automata capture the power of parity automata, and therefore also the power of MSO.

The automaton $\mathcal{A}_\psi$ uses two sets (A) and (B) of additional registers. The idea is that the registers from (A) will be used to verify that the given data tree satisfies ψ; and registers from (B) will be used to verify the content of registers from (A).

The set (A) contains registers $r_{\tau,\sup}$ and $r_{\tau,\inf}$ for every binary MSO-type $\tau(x, y)$ of quantifier-rank at most q. On a node v of an input tree, these registers are intended to store the supremum and infimum data value over all nodes u with $(v, u) \models \tau$, that is

$$\alpha(v, r_{\tau,\sup}) = \sup\{d(u) \mid u \in t \text{ and } (v, u) \models \tau\}$$
$$\alpha(v, r_{\tau,\inf}) = \inf\{d(u) \mid u \in t \text{ and } (v, u) \models \tau\}$$

in an intended register assignment α.

Assuming, that the information stored in registers in (A) is correct, the automaton can locally check that each node satisfies the existential and universal constraints. For instance, for checking existential constraints, the automaton can guess which of the disjuncts is satisfied for each node of the input tree. If it guesses that for a node v there is a node u such that $v <_2 u \wedge \eta_i(v,u)$, then it can verify this guess by checking whether there is a type τ that implies η_i and a node u such that $d(u) > d(v)$ and $(v,u) \models \tau$. The former can be tested via $r_{\tau,\sup}(v) \geq d(v)$.

The values of the registers in (A) are verified with registers from a set (B). The intuition of registers in (B) is as follows. For a node v, the supremum of data values of nodes u with $(v,u) \models \tau$ is either achieved in v itself, or by nodes in $t \setminus t_v$, or in the left subtree $t_{\swarrow v}$ of v, or in the right subtree $t_{\searrow v}$, that is:

$$\alpha(v, r_{\tau,\sup}) = \max \begin{Bmatrix} \sup_u \{d(u) \mid u \in t \setminus t_v \text{ and } (v,u) \models \tau\}, \\ \sup_u \{d(u) \mid u \in t_{\swarrow v} \text{ and } (v,u) \models \tau\}, \\ \sup_u \{d(u) \mid u \in t_{\searrow v} \text{ and } (v,u) \models \tau\} \end{Bmatrix}$$

If $(v,v) \models \tau$, then the maximum is also taken over $d(v)$. The registers in (B) will allow the automaton to deduce the components of the maximum on the right-hand side.

The set (B) contains the following registers for each binary type $\gamma(x,y)$ of quantifier-rank q:

(i) a register $r_{\gamma,\sup,\uparrow}$ intended to store, for a node v, the supremum of all nodes u in $t \setminus t_v$ with $(\uparrow v, u) \models \gamma$,
(ii) a register $r_{\gamma,\sup,\swarrow}$ intended to store, for a node v, the supremum of all nodes u in $t_{\swarrow v}$ with $(\swarrow v, u) \models \gamma$,
(iii) a register $r_{\gamma,\sup,\searrow}$ intended to store, for a node v, the supremum of all nodes u in $t_{\searrow v}$ with $(\searrow v, u) \models \gamma$,

Analogously it contains further registers $r_{\gamma,\inf,\uparrow}$, $r_{\gamma,\inf,\swarrow}$, and $r_{\gamma,\inf,\searrow}$.

Assuming that the information stored in the register from (B) is correct, the automaton can verify the values of registers in (A) using the first equation in Figure 2.

We sketch how the automaton can verify the content of the registers in (B).

For verifying the values of $r_{\tau,\sup,\swarrow}$ and $r_{\tau,\sup,\searrow}$ in a register assignment α, we use an additional registers $r_{\tau,\sup}$ that store, for a node v, the supremum over all nodes u with $(v,u) \models \tau$ in the subtree rooted at v. The values in these registers can be easily "copied" to $r_{\tau,\sup,\swarrow}$ and $r_{\tau,\sup,\searrow}$. The values of $r_{\tau,\sup}$ can now be verified by suprema constraints for a suitable associated language for $r_{\tau,\sup}$ that depends on τ.

The associated language is obtained from the following fact. For every binary MSO-type $t(x,y)$ with free first-order variables x and y there is an alphabet Γ and a regular language L over Γ such that for every tree t there is a labelling with symbols from Γ which is MSO-definable and such that for all pairs (u,v) of nodes: $(u,v) \models t(x,y)$ if and only if the labels on the path from u to v constitute a string from L. This fact is a consequence of the compositionality of MSO-types. The TRASI essentially associates the language L for the MSO-type τ to the register $r_{\tau,\sup}$. As TRASI capture the power of MSO, the automaton can guess and verify the Γ-labeling.

The value of $r_{\tau,\sup,\uparrow}$ on a node v can be verified as follows. Suppose that v is the left child of its parent w (the cases when v is a right child or the root are very similar). Similarly to the reasoning above, the value of $r_{\tau,\sup,\uparrow}$ is either achieved in w, or in $t \setminus t_w$, or in $t_{\searrow w}$. Thus it can be computed using the second equation in Figure 2. This equation can be implemented easily by a tree register automaton with additional auxiliary registers. This concludes the construction of $\mathcal{A}_\psi$.

A tree register automaton with extrema constraints cannot verify that all data values of a tree are distinct. Therefore the automaton $\mathcal{A}_{\neq}$ checks a weaker condition for a register assignment of $\mathcal{A}_\psi$, namely, that if the input value of two nodes v and v' is d then there is a node u on the shortest path from v to v' (which may use tree edges in either direction) such that no register in u has value d. We call an assignment with this property *weakly diverse*. A careful analysis of the proof of decidability of the emptiness problem for tree register automata with extrema constraints shows that if there is an accepting run which is weakly diverse, then there is also a run with distinct data values. Basically, because the values in a subtree t_v can be shifted by π in such a way that fresh data values in registers at node v are distinct from all data values in $t \setminus t_v$.

A TRASI $\mathcal{A}_{\neq}$ can check that the data values of an input data tree for $\mathcal{A}_\psi$ are weakly diverse. ◻

5.2 Satisfiability of $\mathrm{ESO}^2(<_1, <_2)$ reduces to $\mathrm{EMSO}^2(<_1, <_2)$

For a class of structures $\mathcal{K}$ over some fixed signature, we write $\mathrm{ESO}^2(\mathcal{K})$ if we are only interested in evaluating formulas in ESO^2 with structures from $\mathcal{K}$. An $\mathrm{ESO}^2(\mathcal{K})$ formula is satisfiable if it has a model $\mathfrak{A} \in \mathcal{K}$, and we say that such a formula is $\mathcal{K}$-*satisfiable*. Two $\mathrm{ESO}^2(\mathcal{K})$ formulas are equivalent, if they are satisfied by the same structures from $\mathcal{K}$.

For many classes $\mathcal{K}$ of structures the general satisfiability problem of $\mathrm{ESO}^2(\mathcal{K})$ can be reduced to the satisfiability problem of $\mathrm{EMSO}^2(\mathcal{K})$, that is, the fragment of $\mathrm{ESO}^2(\mathcal{K})$ where formulas can only use unary set quantification. Examples are two-variable logic with one linear order [17] and two-variable logic with one relation to be interpreted by a preorder [18].

We show that the same approach works for ordered structures with two linear orders.

Proposition 5.6. *The satisfiability problem of* $\mathrm{ESO}^2(<_1, <_2)$ *reduces to the satisfiability problem of* $\mathrm{EMSO}^2(<_1, <_2)$*, for linear orders* $<_1$ *and* $<_2$.

A general reduction for a broad range of classes of structures is implicit in the procedure used by Pratt-Hartmann in [18]. We make this approach explicit in the following, and then show that it can also be applied to ordered structures with two linear orders. Observe that the satisfiability problems for $\mathrm{ESO}^2(\mathcal{K})$- and $\mathrm{EMSO}^2(\mathcal{K})$-formulas are inter-reducible with the satisfiability problems for $\mathrm{FO}^2(\mathcal{K})$ over arbitrary signatures and, respectively, signatures restricted to unary relation symbols apart from the signature of $\mathcal{K}$. For this reason we restrict our attention to two-variable *first-order logic* in the following, and show that the satisfiability problem for $\mathrm{FO}^2(<_1, <_2)$ over arbitrary signatures effectively reduces to the satisfiability problem of $\mathrm{FO}^2(<_1, <_2)$ with unary relations, for linear orders $<_1$ and $<_2$.[1]

Now fix a class $\mathcal{K}$ of structures over some signature Δ. An $\mathrm{FO}^2(\mathcal{K})$ formula is in *spread normal form* [18, p. 23] if it conforms to the pattern

$$\bigwedge_{\theta \in Z} \exists x \theta(x) \wedge \forall x \forall y (x = y \vee \nu(x, y))$$
$$\wedge \quad \bigwedge_{k=0}^{2} \bigwedge_{h=0}^{m-1} \forall x \exists y (\lambda_k(x) \rightarrow (\lambda_{\lfloor k+1 \rfloor}(y) \wedge \mu_h(y) \wedge \theta_h(x, y)))$$

where (i) Z is a set of unary pure Boolean formulas; (ii) ν, $\theta_0, \ldots, \theta_{m-1}$ are quantifier- and equality-free formulas; and (iii) λ_0, λ_1, and λ_2 are mutually exclusive unary pure Boolean formulas; and (iv) $\mu_0, \ldots, \mu_{m-1}$ are mutually exclusive unary pure Boolean formulas. Here, a formula is said to be *pure Boolean* if it is quantifier-free and does not use symbols from Δ, and $\lfloor k+1 \rfloor$ denotes $k+1 \pmod 3$. Thus, as stated by Pratt-Hartmann, "the essential feature of formulas in spread normal form is that witnesses are 'spread' over disjoint sets of elements".

We say that $\mathrm{FO}^2(\mathcal{K})$ has a spread normal form if for every $\mathrm{FO}^2(\mathcal{K})$ formula φ there is a $\mathrm{FO}^2(\mathcal{K})$ formula φ^* in spread normal form such that φ has a (finite) model if and only if φ^* has a (finite) model.

The proofs of Lemma 4.3, Lemma 4.4, and Theorem 4.5 in [18] implicitly imply the following lemma.

Lemma 5.7. *Suppose $\mathcal{K}$ is a class of structures with signature Δ such that $\mathrm{FO}^2(\mathcal{K})$ has a spread normal form. Then from an $\mathrm{FO}^2(\mathcal{K})$ formula over signature $\Delta \uplus \Theta$ one can effectively construct an equisatisfiable $\mathrm{FO}^2(\mathcal{K})$ formula over signature $\Delta \uplus \Theta'$ where Θ' only contains unary relation symbols.*

In the following we describe a general criterion for establishing that $\mathrm{FO}^2(\mathcal{K})$ over arbitrary signatures has a spread normal form. This criterion generalizes the argument used by Pratt-Hartmann for establishing a spread normal form for partial orders. Intuitively, a spread normal form exists, if non-King elements of a structure that satisfies a $\mathrm{FO}^2(\mathcal{K})$ formula can be cloned without affecting the truth of the formula. Recall that an element is a *King*, if its quantifier-free type is unique within the structure.

[1] We prefer to state results in terms of ESO^2 and EMSO^2 instead of FO^2 as the abbreviation FO^2 is ambiguously used for two-variable logic over unary and arbitrary signatures in the literature.

Suppose $\mathcal{K}$ is a class of structures over Δ, and $\mathcal{A}$ is an expansion of a $\mathcal{K}$-structure with domain A over a signature $\Delta \uplus \Theta$. Then $\mathcal{A}$ *allows cloning* of a set of non-King elements B, if there is a structure $\mathcal{A}'$ with domain $A' \stackrel{df}{=} A \cup B'$ where $B' = \{b' \mid b \in B\}$ such that

- (i) $\mathcal{A} \subseteq \mathcal{A}'$ and all quantifier-free binary types realized in $\mathcal{A}'$ are realized in $\mathcal{A}$;
- (ii) $\mathrm{TP}^{\mathcal{A}'}[a, b'] = \mathrm{TP}^{\mathcal{A}}[a, b]$ for all $a, b \in A$ with $a \neq b$; and
- (iii) $\mathrm{TP}^{\mathcal{A}'}[b_1', b_2'] = \mathrm{TP}^{\mathcal{A}}[b_1, b_2]$ for all $b_1, b_2 \in A$ with $b_1 \neq b_2$; and
- (iv) the Δ-structure obtained from $\mathcal{A}'$ by forgetting the relations in Θ is a $\mathcal{K}$-structure.

Here, $\mathrm{TP}^{\mathcal{A}}[a, b]$ denotes the quantifier-free type of the tuple (a, b). The class $\mathcal{K}$ allows cloning if every expansion of a $\mathcal{K}$-structure allows cloning of arbitrary sets of non-King elements.

Lemma 5.8 (Adaption of Lemma 4.2 in [18])**.** *If $\mathcal{K}$ allows cloning then $\mathrm{FO}^2(\mathcal{K})$ has a spread normal form.*

The final ingredient for the proof of Proposition 5.6 is that structures with two linear orders allow cloning.

Lemma 5.9. *The class of structures with two linear orders allows cloning.*

Proof sketch. Consider a structure $\mathcal{A}$ with signature $\{<_1, <_2\} \uplus \Theta$ and domain A that interprets $<_1$ and $<_2$ by linear orders. Let B be a set of non-King elements of $\mathcal{A}$. The structure $\mathcal{A}'$ that clones B in $\mathcal{A}$ has domain $A' \stackrel{df}{=} A \cup \{b' \mid b \in B\}$. The simple idea is to insert a copy b' of $b \in B$ immediately before/after b with respect to $<_1$ and $<_2$. If $c \in A$ is a non-king with the same unary type as b (such an element exists since b is a non-King) and if, for instance, $b <_1 c$ and $b <_2 c$ then b' is inserted directly after b in both orders $<_1$ and $<_2$. The type of (b, b') is then inherited from (b, c). Furthermore the element b' inherits its relation to all other elements from b'. □

We can now prove Proposition 5.6. By Lemma 5.9 structures with two linear order allow cloning, and therefore $\mathrm{FO}^2(<_1, <_2)$ has spread normal form by Lemma 5.8. Now Proposition 5.6 follows from Lemma 5.7.

Acknowledgments

We are grateful to Nathan Lhote for insightful discussions, in particular on the inclusion of MSO-definable atoms in Theorem 5.1. We also thank the anonymous reviewers for helpful suggestions on improving the readability of the paper.

References

[1] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. 2008. *Principles of model checking.* MIT press.

[2] Achim Blumensath, Thomas Colcombet, and Christof Löding. 2008. Logical theories and compatible operations. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas].* 73–106.

[3] Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. 2009. Two-variable logic on data trees and XML reasoning. *J. ACM* 56, 3 (2009), 13:1–13:48. https://doi.org/10.1145/1516512.1516515

[4] Witold Charatonik and Piotr Witkowski. 2013. Two-Variable Logic with Counting and Trees. In *LICS'13*. 73–82.

[5] Luc Dartois, Emmanuel Filiot, and Nathan Lhote. 2018. Logics for Word Transductions with Synthesis. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, Anuj Dawar and Erich Grädel (Eds.). ACM, 295–304. https://doi.org/10.1145/3209108.3209181

[6] Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. 1997. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic* 3, 1 (1997), 53–69.

[7] Erich Grädel and Martin Otto. 1999. On Logics with Two Variables. *Theor. Comput. Sci.* 224, 1-2 (1999), 73–113.

[8] Jean Christoph Jung, Carsten Lutz, and Thomas Zeume. 2019. Decidability and Complexity of ALCOIF with Transitive Closure (and More). In *Proceedings of the 32nd International Workshop on Description Logics, Oslo, Norway, June 18-21, 2019 (CEUR Workshop Proceedings)*, Mantas Simkus and Grant E. Weddell (Eds.), Vol. 2373. CEUR-WS.org. http://ceur-ws.org/Vol-2373/paper-17.pdf

[9] Michael Kaminski and Nissim Francez. 1994. Finite-Memory Automata. *Theor. Comput. Sci.* 134, 2 (1994), 329–363. https://doi.org/10.1016/0304-3975(94)90242-9

[10] Emanuel Kieronski. 2011. Decidability Issues for Two-Variable Logics with Several Linear Orders. In *CSL 2011 (LIPIcs)*, Vol. 12. 337–351.

[11] Emanuel Kieronski and Martin Otto. 2012. Small substructures and decidability issues for first-order logic with two variables. *J. Symb. Log.* 77, 3 (2012), 729–765.

[12] Emanuel Kieronski and Lidia Tendera. 2009. On Finite Satisfiability of Two-Variable First-Order Logic with Equivalence Relations. In *LICS 2009*. IEEE Computer Society, 123–132.

[13] Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer.

[14] Michael Mortimer. 1975. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math.* 21 (1975), 135–140.

[15] Frank Neven. 2002. Automata theory for XML researchers. *ACM Sigmod Record* 31, 3 (2002), 39–46.

[16] Frank Neven, Thomas Schwentick, and Victor Vianu. 2004. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.* 5, 3 (2004), 403–435. https://doi.org/10.1145/1013560.1013562

[17] Martin Otto. 2001. Two Variable First-Order Logic over Ordered Domains. *J. Symb. Log.* 66, 2 (2001), 685–702.

[18] Ian Pratt-Hartmann. 2018. Finite satisfiability for two-variable, first-order logic with one transitive relation is decidable. *Math. Log. Q.* 64, 3 (2018), 218–248. https://doi.org/10.1002/malq.201700055

[19] Michael Oser Rabin. 1972. *Automata on Infinite Objects and Church's Problem*. Number 13. American Mathematical Soc.

[20] Thomas Schwentick and Thomas Zeume. 2012. Two-Variable Logic with Two Order Relations. *Logical Methods in Computer Science* 8, 1 (2012).

[21] Dana Scott. 1962. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic* 27, 377 (1962), 74.

[22] Saharon Shelah. 1975. The monadic theory of order. *Annals of Mathematics* (1975), 379–419. https://doi.org/10.2307/1971037

[23] Wolfgang Thomas. 1997. Languages, automata, and logic. In *Handbook of formal languages*. Springer, 389–455.

[24] Thomas Zeume and Frederik Harwath. 2016. Order-Invariance of Two-Variable Logic is Decidable. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 807–816. https://doi.org/10.1145/2933575.2933594