REGISTER AUTOMATA WITH EXTREMA CONSTRAINTS, AND AN APPLICATION TO TWO-VARIABLE LOGIC

SZYMON TORUŃCZYK AND THOMAS ZEUME

University of Warsaw
e-mail address: szymtor@mimuw.edu.pl

Ruhr University Bochum
e-mail address: 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.

1. INTRODUCTION

Automata for words and trees find applications in diverse areas such as logic, verification, and database theory (see, e.g., [Tho97, BKL08, Nev02]). 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* [KF94, NSV04].

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 [Mor75, Sco62]. 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.

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 [Pra18]. Previously ESO^2 with two equivalence relations [KO12, KT09] and ESO^2 with two "forest" relations have been shown to be decidable [CW13]. 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 [SZ12, ZH16]. The satisfiability problem and the finite satisfiability problem for EMSO^2 is undecidable for three linear orders [Kie11].

The question whether two-variable logic with two linear orders is decidable for general (not necessarily finite) structures has been left open in [SZ12, ZH16], 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 [DFL18] and in recent work by Lutz, Jung, and Zeume in relation to description logics [JLZ20].

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 [BMSS09].

An immediate consequence of Theorem 1.2 is the decidability of satisfiability of ESO^2 with two linear orders.
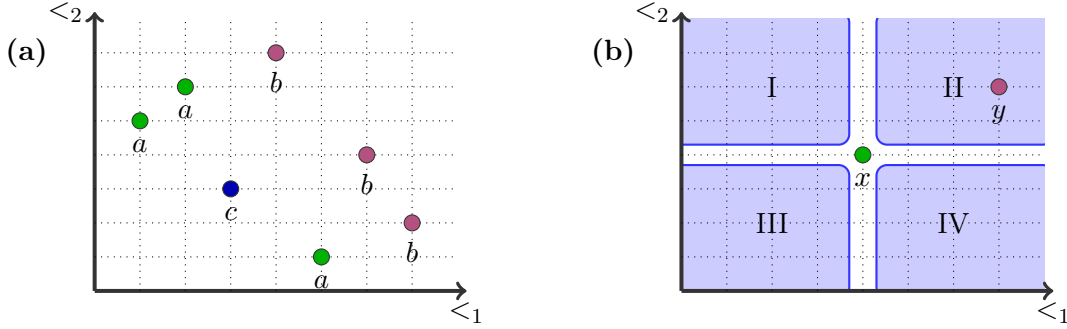
FIGURE 1. **(a)** *A structure with two linear orders represented as a point set in the two-dimensional plane. The structure satisfies the existential constraint* $\forall x \exists y (a(x) \to (b(y) \land x <_1 y \land x <_2 y))$, *and the universal constraint* $\forall x \forall y \neg (a(x) \land b(y) \land y <_1 x \land y <_2 x)$. *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* $x <_1 y \land x <_2 y$.

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 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* [SZ12]. 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 [SZ12]. 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 [Rab72].

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 $<_{\mathrm{anc}}$, 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.

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 $\overline{\mathbb{R}} \stackrel{df}{=} \mathbb{R} \cup \{-\infty, \infty\}$ 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* $\mathbf{D} \stackrel{df}{=} \langle \overline{\mathbb{R}}, < \rangle$ 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 *states* Q;
- a finite set of *root states* $F \subseteq Q$;
- a finite set of *input registers* I;
- 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 over Σ (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 a *rank* to each state.

If a TRASI does not use suprema and infima terms ($\sup r$ or $\inf r$ for $r \in R$) in its transition relation 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 itself) 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 $r_\rho(v)$ for the value of register r in pre-run ρ, or $r(v)$ 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:

- $\sigma, \swarrow\sigma$, 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 $r(v)$. A term of the form $\swarrow r$, for $r \in R$, evaluates to the value $r(\swarrow v)$, and likewise for $\searrow r$. Finally, a term of the form $\sup r$ evaluates to $r_{\sup}(v)$, and a term of the form $\inf r$ evaluates to $r_{\inf}(v)$.
- 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, $r_{\sup}(v)$ is the supremum of the values $r(w)$, where w ranges over all L-descendants of v where L is the language associated to r, whereas $r_{\inf}(v)$ 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 [Rab72, Tho97].

Example 3.1. (a) *Consider the language of trees with one data value per node such that the values on each infinite path form a strictly increasing sequence of numbers. A TRASI for this language can work as follows. 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.

(b) *Consider the language of trees over $\Sigma = \{a, b, c\}$ with one data value per node where the data value of each a-labeled node equals the supremum of the b-labeled nodes below. A TRASI for this language can work as follows. The data value is stored in an input register r with associated language $\Sigma^* b$. The automaton uses two states $q_\top$ and $q_\bot$, where $q_\top$ represents the situation where all b-nodes seen on a path so far adhere to the condition. A path is accepted if $q_\top$ occurs infinitely often.*

The transition relation is defined by

$$\left(\swarrow q_\top \leftrightarrow (q_\top \wedge (\neg a \vee (r = \sup r)))\right) \wedge \left(\searrow q_\top \leftrightarrow (q_\top \wedge (\neg a \vee (r = \sup r)))\right).$$

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 Sections 4.1 and 4.2) 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 4.3), this will yield decidability of the emptiness problem for TRASI.

4.1. **Warm-Up: From TRASI with Σ^* to Tree Register Automata.** For the sake of readability, we first 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 following section. In this section, we henceforth assume that all languages associated with registers in a TRASI are equal to Σ^*.

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. Fix an input tree t, a register r and a node v; recall that t_v is the set of descendants of v in t. If the supremum of the values of r in t_v is equal to c, then either the supremum is not attained at some node w of t_v, or there must be an infinite path starting from v such that the supremum of r has value c for all nodes along this path (cf. Fig. 2). 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 eventually exceed any values smaller than c which are stored at any point along the path in some register. 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 $\swarrow r < \searrow s$ 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.*
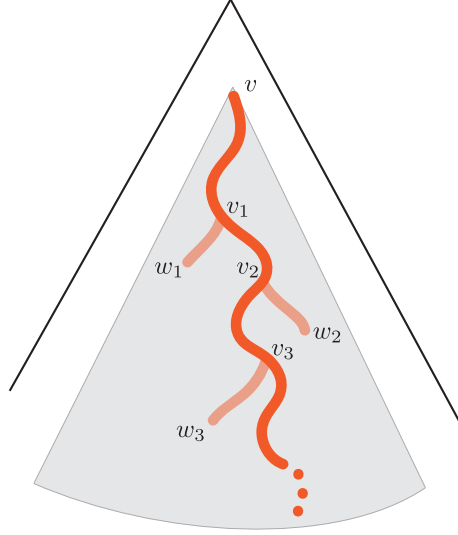
FIGURE 2. *An infinite path and finite paths departing from it, forming a witness of a supremum. The path is constructed so that $\sup\{r(w) \mid w \in t_{v_n}\}$ is the same for all $n \geq 0$, and equal to $\sup\{r(w_n) \mid n \geq 1\}$.*

The existence of a pre-run as described 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.

Intuitively, a witness family partitions the nodes of a tree into finite paths such that each such path provides a witness for an extremum of some register. To this end, paths of such a partition are labeled by r_+ or r_-, indicating whether they are the supremum or infimum witness for a register r. This is formalized as follows.

We introduce two symbols r_+ and r_- for each register $r \in R$. Let $R^\pm = \{r_+, r_- \mid r \in R\}$ be the collection of all such symbols, and fix an arbitrary total order on $R^\pm$. Further fix a partition $\mathcal{P}$ of the set of nodes of the tree underlying a pre-run ρ into finite paths (cf. Fig. 3). Say that a path π in $\mathcal{P}$ is the *father* of a path π' if the father of the smallest node (wrt. the ancestor order) in π' belongs to π.

Let $\lambda\colon \mathcal{P} \to R^\pm$ be a labeling of the paths in $\mathcal{P}$. A path $\pi \in \mathcal{P}$ is called a sup-*witness path* associated to r if $\lambda(\pi) = r_+$ and it is called a inf-*witness path* associated to r if $\lambda(\pi) = r_-$. 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
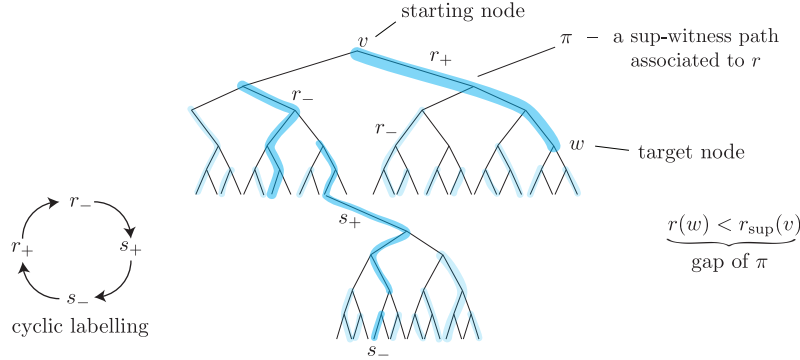
FIGURE 3. *A partition of the tree into witness paths, a labelling of the paths with*
$R^\pm$ *and the gap of a witness path.*

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
$\lambda(\pi) = r_+$ then the target value of π is at most equal to the associated extremum of π, since
ρ is extrema-consistent (if $\lambda(\pi) = r_-$, 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. Intuitively, the length of the gap is a measure of the quality of the
witness path, shorter gaps being better witnesses.

Our goal is to state properties of a partition $\mathcal{P}$ of a pre-run and a labeling $\lambda \colon \mathcal{P} \to R^\pm$
that guarantee the existence of a run (i.e., a pre-run that is consistent with respect to the
extrema). Such a combination of a partition and a labeling will be a certificate for the
existence of a run. For the case where all associated languages are Σ^*, periodically witnessing
extrema for each register with close enough target values suffices as a certificate.

A mapping $\lambda \colon \mathcal{P} \to R^\pm$ to labels from an ordered set $R^\pm$ is a *cyclic labeling* if:

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

Clearly, there is exactly one cyclic labeling of $\mathcal{P}$ with labels from $R^\pm$.

Henceforth (in this section), when considering a partition $\mathcal{P}$ of a partial run ρ, we assume
that each path in $\mathcal{P}$ is labeled according to the cyclic labeling with labels from $R^\pm$ assuming
an arbitrary order on $R^\pm$.

We now define when a partition $\mathcal{P}$ 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 for the case when all associated languages are trivial.

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 $\pi \in \mathcal{P}$ 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. $\qquad \square$

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 $f \colon \mathbf{D} \to \mathbf{D}$ 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,

$$r_{\sup}(v) = \sup\{r(v') \mid v' \text{ is a descendant of } v\} \tag{4.1}$$

$$r_{\inf}(v) = \inf\{r(v') \mid v' \text{ is a descendant of } v\}. \tag{4.2}$$

We prove (4.1) while (4.2) is proved analogously. The inequality $\geq$ in (4.1) follows from extrema-consistency of ρ'.

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 (4.1) is equal to $r(w)$, which proves (4.1) as $r_{\sup}(v) = r(w)$.

Suppose now that π is infinite. To prove (4.1), it suffices to exhibit a sequence $w_1, w_2, \ldots$ of descendants of v such that

$$r_{\sup}(v) = \lim_{n \to +\infty} r(w_n). \tag{4.3}$$

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, as $\mathcal{P}$ is a partition into finite paths, and the 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:

$$|r(w_n) - r_{\sup}(v)| = |r(w_n) - r_{\sup}(u_n)| < \frac{1}{n}. \tag{4.4}$$

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 (4.4) holds for the pre-run ρ' and for all n, this proves (4.3), yielding (4.1). Hence ρ' is a run. $\qquad\square$

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. $\qquad\square$

The special case of Theorem 4.1 where all associated languages are Σ^* now follows easily via 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 Section 4.3.

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. **From TRASI to Tree Register Automata.**
In this section, we lift the construction from the previous section to TRASIs where languages associated with registers can be arbitrary regular languages.

Suppose that $\mathcal{A}$ is a TRASI. As before, we intend to construct a tree register automaton $\widehat{\mathcal{A}}$ that checks conditions on a pre-run that ensure that there is an actual run of $\mathcal{A}$. Again, we assume that the TRASI $\mathcal{A}$ is normalized.

We first adapt extrema-consistency for the case of arbitrary associated regular languages. Consider a register r with associated language L. If the supremum of r at a σ-node v is c

then either (1) $\varepsilon \in L$ and r has value c at node v, or (2) c is the supremum of the r-values of L'-descendants of the left child of v, where $L' \stackrel{df}{=} \{x \mid cx \in L\}$, or (3) likewise for the right child of v. It will therefore be convenient to assume that there are registers with such associated languages L'.

For a regular language L there are only finitely many languages $u^{-1}L \stackrel{df}{=} \{x \mid ux \in L\}$ for $u \in \Sigma^*$. We say that $\mathcal{A}$ is *language-closed* if whenever a register r has L as associated language, then there is also a register with associated language $u^{-1}L$ for all $u \in \Sigma^*$. If r is a register with associated language L, then we denote by ur the register with associated language $u^{-1}L$.

From now on, we assume without loss of generality that $\mathcal{A}$ is language-closed.

Let ρ be a pre-run of $\mathcal{A}$. Say that ρ is *regular extrema-consistent* if for every node v labeled with $\sigma \in \Sigma$ and $r \in R$ with associated language L,

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

where $d_{\sup} \stackrel{df}{=} r(v)$ if $\varepsilon \in L$ and $d_{\sup} = -\infty$ otherwise, and similarly for $d_{\inf}$.

Clearly, every run of $\mathcal{A}$ is regular 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 regular extrema-consistent.*

Next we adapt certificates. We introduce witness families that ensure that for each node v and each register r with associated language L, the supremum of r at v is approached by values of r of L-descendants of v. Essentially, again, this will be certified by a path starting from v on which there is a sequence of nodes $v_1, v_2, \ldots$ with nodes $w_1, w_2, \ldots$ below them (but not necessarily on the path) such that (1) there is an L-path from v to all w_i and (2) the r-values of the w_i tend to c. The paths from v_i to w_i will be paths in the witness family. The existence of such witness families will be verifiable by tree register automata (without suprema constraints).

A complication arises from using arbitrary regular languages. Suppose that $\mathcal{B} = (Q, \delta, s, F)$ is a deterministic finite state automaton for L. If a path from v to v_i brings $\mathcal{B}$ from its initial state to state q, then the path from v_i to w_i has to be labeled by a word in the language $L(B_q)$ of $B_q \stackrel{df}{=} (Q, \delta, q, F)$ to ensure that the path from v to w_i is in L. Furthermore, the extrema of all nodes v along a path π have to be witnessed as, intuitively, the automaton $\mathcal{B}$ is "started" at each node $v \in \pi$.

As in the previous section, our goal is to state properties of a partition $\mathcal{P}$ of a pre-run of our TRASI $\mathcal{A}$ and a labeling $\lambda \colon \mathcal{P} \to R^\pm$ that guarantee the existence of a run of $\mathcal{A}$ on the same input tree (i.e., a pre-run that is consistent with respect to the suprema). For the case where all associated languages may be arbitrary regular languages, the labeling is slightly more involved and depends on the associated languages.

For ensuring witnesses for a register r with associated language L for all nodes, we consider trees that encode all runs of an automaton for L along paths of an input tree. Formally, let $\mathcal{B} = (Q, \delta, s, F)$ be a minimal deterministic finite state automaton and $<$ an arbitrary order on Q. Let $|Q| = m$; the elements of $\{1, \ldots, m\}$ will be called *tapes*. The *run tree* $\widehat{t}$ of a tree t, defined below, associates to each tape and node v a state in Q. The idea
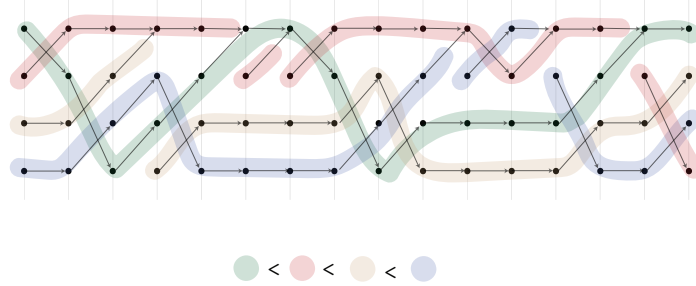
FIGURE 4. *Construction of tapes along an input word. The automaton has 4 states and is deterministic. The dots are pairs consisting of a position in the word and a state of the automaton. The arrows denote the transitions of the automaton along the input letters. The colors correspond to the tapes. Every color occurs exactly once at each position, and when following an edge, the color can either remain unchanged or decrease.*

is that tapes encode runs of $\mathcal{B}$ along the path. However, whenever two tapes $i < j$ enter the same state, only the smaller tape i continues this run; the other tape starts a new run from an "unused" state. See Fig. 4 for the construction of the run tree in the case when t is a word. The construction for trees is performed analogously, by carrying out the same construction on each branch.

Formally the *run tree* $\widehat{t}$ of $\mathcal{B}$ for a tree t associates a tuple $\bar{q} \in Q^m$ to each node v of t as follows. The tuples $\bar{q}$ will always have all coordinates distinct. The root is annotated by $(q_1, \ldots, q_m)$ with $q_i < q_{i+1}$. If a node is labeled by $\sigma \in \Sigma$ and annotated by $\bar{q} = (q_1, \ldots, q_m) \in Q^m$ then its children are annotated by the permutation $\bar{p} = (p_1, \ldots, p_m)$ obtained as follows. Let $\bar{p}' = (p'_1, \ldots, p'_m)$ be such that $p'_i = \delta(q_i, \sigma)$. Then $\bar{p}$ is constructed from $\bar{p}'$ by only keeping the smallest copies of identical states and replacing the other copies by states not used in $\bar{p}'$ in increasing order. Formally the components of $\bar{p}$ are defined inductively for $i = 1, \ldots, m$:

- if p'_i is not equal to $p_1, \ldots, p_{i-1}$ then $p_i \stackrel{df}{=} p'_i$,
- else p_i is defined to be the smallest state not occuring in $\bar{p}'$ and $p_1, \ldots, p_{i-1}$.

The *run tree of a register r* is the run tree of the minimal deterministic finite state automaton for the language associated with r.

We now state properties of a partition $\mathcal{P}$ of a pre-run of our TRASI $\mathcal{A}$ and a labeling $\lambda \colon \mathcal{P} \to R^{\pm}$ that guarantee the existence of a run on the same input tree.

Intuitively, a regular labeling $\lambda \colon \mathcal{P} \to R^{\pm}$ ensures that all "parallel" runs of an automaton for a language associated to some register are witnessed periodically. Formally, let

$$K^{\pm} \stackrel{df}{=} \{(r^+, i), (r^-, i) \mid r \in R, i \in \{1, \ldots, m\}\}.$$

A labeling $\lambda \colon \mathcal{P} \to R^{\pm}$ is *regular* if it is obtained from a cyclic labeling $\kappa \colon \mathcal{P} \to K^{\pm}$ as follows. Consider a path π with $\kappa(\pi) = (r_+, i)$ and smallest node v. Suppose that the run tree of r annotates v with $(q_1, \ldots, q_m)$, then $\lambda(\pi)$ is t_+ where t is the register with associated language $L(\mathcal{B}_{q_i})$. Analogously for $\kappa(\pi) = (r_-, i)$. Intuitively, the path π is a witness for all ancestors of v for which the automaton for the language associated with r reaches state q_i at node v.

Henceforth (in this section), when considering a partition $\mathcal{P}$ of a partial run ρ, we assume that each path in $\mathcal{P}$ is labeled according to the (unique) regular labeling with labels from $R^{\pm}$.

We now define when a partition $\mathcal{P}$ is a regular certificate. The partition $\mathcal{P}$ is a *regular 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 $s \in \mathcal{R}$.

As previously, we prove the following three properties of regular certificates:

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

These properties together easily yield Theorem 4.1. Their proofs are adaptions of the proofs Lemmata 4.2, 4.3 and 4.4 from the previous section. For the sake of completeness, we repeat the essential proof steps.

Lemma 4.5. *Every run of $\mathcal{A}$ has a regular 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 $k \in K^{\pm}$ be the smallest label in $K^{\pm}$. Otherwise, the parent of v already belongs to some path $\pi \in \mathcal{P}$ which has an assigned label k', and let $k \in K^{\pm}$ be the successor of k' in cyclic order, according to the fixed order on $K^{\pm}$. Assume $k = (r_{+}, i)$ for some $r \in R$. The case when $k = (r_{-}, i)$ is treated symmetrically. Suppose that the run tree of r annotates v with $(q_1, \ldots, q_m)$, then l is the label t_{+} where t is the register with associated language $L(\mathcal{B}_{q_i})$. Pick any descendant w of v such that the interval with endpoint $t(w)$ (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 regular certificate. $\square$

Lemma 4.6. *For every (locally consistent, regular extrema-consistent) pre-run ρ which has a regular 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 regular certificate $\mathcal{P}$. A run ρ' which has the same states as ρ is constructed exactly as in the proof of Lemma 4.3.

The difference with the proof of Lemma 4.3 is in showing that ρ' is a run, i.e., that for every node v and register r with associated language L,

$$r_{\sup}(v) = \sup\{r(v') \mid v' \text{ is an } L\text{-descendant of } v\} \tag{4.5}$$

$$r_{\inf}(v) = \inf\{r(v') \mid v' \text{ is an } L\text{-descendant of } v\}. \tag{4.6}$$

We prove (4.5) while (4.6) is proved analogously. The inequality $\geq$ in (4.5) follows from the fact that ρ' is extrema-consistent.

Denote by $\vartheta(v, v')$ the word between the nodes v and v'. Recall that $\mathcal{A}$ is language-closed and therefore for every register r with associated language L there is a register ϑr with associated language $\vartheta^{-1} L$, for all $\vartheta \in \Sigma^{*}$.

Towards proving $\leq$, pick an inclusion-maximal branch π starting at v, such that $r_{\sup}(v) = \vartheta(v, v') r_{\sup}(v')$ for all $v' \in \pi$. If the branch π is finite then $r_{\sup}(v) = \vartheta(v, w) r(w)$ where w is the largest node in π (wrt. the ancestor order) and $\sigma r_{\sup}(\swarrow w), \sigma r_{\sup}(\searrow w)$ are both smaller

than $r(w)$ and σ is the label of w. By regular extrema-consistency of ρ' it follows that the right-hand side in (4.5) is equal to $\vartheta(v,w)r(w)$, which proves (4.5) as $\vartheta(v,w)r_{\sup}(v) = r(w)$.

Suppose now that π is infinite. To prove (4.5), it suffices to exhibit a sequence $w_1, w_2, \ldots$ of descendants of v such that

$$r_{\sup}(v) = \lim_{n \to +\infty} r(w_n). \tag{4.7}$$

Construct sup-witness paths $\pi_0, \pi_1, \ldots$ starting at nodes $v = u_0, u_1, \ldots$ associated to registers $\vartheta(v, u_i)r$, 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 $\vartheta(v, u_n)r$ 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 regular.

Then the nodes $v = u_0, u_1, \ldots$ all lie on the path π. In particular, $r_{\sup}(v) = \vartheta(v, u_n)r_{\sup}(u_n)$ 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:

$$|\vartheta(v, w_n)r(w_n) - r_{\sup}(v)| = |\vartheta(v, w_n)r(w_n) - r_{\sup}(u_n)| < \frac{1}{n}. \tag{4.8}$$

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 (4.8) holds for the pre-run ρ' and for all n, this proves (4.7), yielding (4.5). Hence ρ' is a run. $\square$

Lemma 4.7. *There is a register automaton $\mathcal{B}$ which accepts a pre-run ρ of $\mathcal{A}$ if and only if it has a regular certificate.*

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 from $K^{\pm}$ and from $R^{\pm}$ and the automaton verifies that nodes in the same part of $\mathcal{P}$ have equal labels, and that this yields a regular labeling of $\mathcal{P}$ by simulating the finite state automata for the languageas associated to registers. $\square$

As previously, the properties (1)-(3) listed above and which have now been proved, yield a reduction from the emptiness problem for TRASI to the emptiness problem for tree register automata. It remains to show that the latter problem is decidable.

4.3. 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 4.8. *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* $\text{TP}(v)$ of v is the set of all satisfied propositional formulas with variables of the form $\sigma, \swarrow\sigma$, 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* $\text{TP}(t)$ over Σ_{Types} as follows. The tree $\text{TP}(t)$ has the same set of nodes as t and each node v is labelled by $\text{TP}(v)$.

Lemma 4.9. *One can construct a parity tree automaton $\mathcal{A}'$ over Σ_{Types} such that for any tree t over $\Sigma \times \mathbf{D}^I$,*

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

Proof. This follows from the fact that whether $\mathcal{A}$ accepts t depends only on the formulas satisfied by each triple of neighbouring nodes. $\qquad\square$

Lemma 4.10. *One can construct a parity automaton $\mathcal{B}'$ over Σ_{Types} such that for every tree t' over Σ_{Types}:*

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

Proof sketch. The automaton $\mathcal{B}'$ only checks for consistency of neighbouring labels in t'. Before explaining why this suffices, we define this notion more precisely.

We say that a triple of labels $\tau, \swarrow\tau, \searrow\tau \in \Sigma_{\text{Types}}$ is *consistent* if there exists a finite binary tree t over $\Sigma \times \mathbf{D}^I$ such that in $\text{TP}(t)$ the root and its two sons have labels $\tau, \swarrow\tau, \searrow\tau$, respectively. The tree t is called a witness for $\tau, \swarrow\tau$, and $\searrow\tau$.

Note that it suffices to consider witnesses which are binary trees of height 2. Therefore, testing whether a triple of labels $\tau, \swarrow\tau, \searrow\tau$ is consistent amounts to testing satisfiability of a formula over constantly many variables $\Sigma \times \mathbf{D}^I$. It boils down to testing if propositional variables that encode inequalities of register values are consistent. For example, if τ states that two registers of the right child compare in a certain way, say $\searrow s < \searrow t$ for two registers s and t, then $\searrow\tau$ must agree with that, i.e. it must state that $s < t$. This discussion implies that set of consistent triples is computable. Obviously, it is also finite, since Σ_{Types} is finite.

The automaton $\mathcal{B}'$ checks that for every node v with children $\swarrow v$ and $\searrow v$, the triple of labels of these nodes is consistent.

Clearly, if $t' = \text{TP}(t)$ for some tree t over $\Sigma \times \mathbf{D}^I$, then t' is accepted by $\mathcal{B}'$. It therefore remains to show that if a tree t' is accepted by $\mathcal{B}'$, then there exists a tree t over $\Sigma \times \mathbf{D}^I$ such that $t = \text{TP}(t')$. We construct the tree t in a top-down fashion, defining the labels of the nodes of t by induction on their height. In each step, we use the values provided by the witnesses, and deform them using density of $\mathbf{D}$ so that they fit well into the values of t constructed so far. $\qquad\square$

Proof (of Lemma 4.8).   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 $t = \text{TP}(t')$ by Lemma 4.10 which is accepted by $\mathcal{A}$ due to Lemma 4.9. The other direction is immediate.

Since parity automata are closed under intersection, there is a parity tree automaton $\mathcal{C}'$ 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 $\mathcal{C}'$. $\square$

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.

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, from now on and without loss of generality, that all relation symbols are of arity at most two; see [GKV97, page 5] for a justification. Restricting the set quantifiers of ESO^2 to be unary yields the fragment EMSO^2.

Our interest is in the satisfiability problems for ESO^2 and EMSO^2 with two orders $<_1$ and $<_2$. It is easy to see that two-variable logics cannot express that a binary relation is transitive and in particular that it is an order. Therefore we study two-variable logics, where certain relation symbols are supposed to be interpreted only by order relations. More formally, for a class $\mathcal{K}$ of structures over some fixed signature Δ, we write $\text{ESO}^2(\mathcal{K})$ if we are only interested in evaluating formulas in ESO^2 with structures from $\mathcal{K}$. An $\text{ESO}^2(\mathcal{K})$ 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 $\text{ESO}^2(\mathcal{K})$ formulas are equivalent, if they are satisfied by the same structures from $\mathcal{K}$.

By $\text{ESO}^2(<_1, <_2)$ and $\text{EMSO}^2(<_1, <_2)$ we denote that $\Delta = \{<_1, <_2\}$ where $<_1$ and $<_2$ are interpreted as orders. Often we will be interested in orders of a certain shape only, which we will then state explicitly. For instance, when saying "$\text{ESO}^2(<_1, <_2)$ for a tree order $<_1$ and a linear order $<_2$", we mean ESO^2 formulas with two distinguished binary relation symbols $<_1$ and $<_2$ which are always interpreted by a tree order and a linear order. For simplicity, we assume that domains always have size at least two and we often identify relation symbols with their respective interpretations.

Let $\text{EMSO}^2(\text{MSO}(<_1), <_2)$ denote the set of $\text{EMSO}^2(<_1, <_2)$ formulas that can use binary atoms definable in $\text{MSO}(<_1)$ 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 $\text{EMSO}^2(\text{MSO}(<_1), <_2)$, 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 $\text{EMSO}^2(\text{FO}(<_1), <_2)$, i.e., for the logic where the additional atoms are restricted to be definable in FO. This follows from the observation that (general) 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. $\square$

Using techniques developed by Pratt-Hartmann in [Pra18] 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 5.3 yields the following result.

Corollary 5.4. *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 of Theorem 5.1 is a similar decidability result, for the the special case when $<_1$ is restricted to be the ancestor order of the complete binary tree (see Section 5.1.1). 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 (see Section 5.1.2).

5.1.1. *Countable satisfiability for* $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$ *in the Full Binary Tree.* In this section the focus is on deciding $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$ in the full binary tree.

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.*

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. Then Proposition 5.5 follows immediately as emptiness for TRASIs is decidable.

Instead of constructing $\mathcal{A}$ from a general $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$ sentence φ, it will be convenient to start from a normal form for such formulas. Say that an $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$-formula is in *Scott normal form*, if it adheres to the form

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

where ψ and all ψ_i are quantifier-free formulas whose atoms are $\mathrm{MSO}(<_1)$-formulas and relation symbols from $\bar{Z} \cup \{<_2\}$. It is well known that a $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$-formula can

be effectively transformed into an equisatisfiable $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$-formula in Scott normal form (see, e.g., [Sco62] and [GO99, page 17]).

Under the assumption that $<_2$ is a linear order, formulas in Scott normal form can be further simplified. An $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$-formula is in *constraint normal form* if it is of the form $\exists \bar{Z} \psi$ 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, quantifier-free $\mathrm{MSO}(<_1)$ formulas with free variables x and y which may use relation symbols from $\bar{Z}$.

Formulas in Scott normal form can be easily converted into equisatisfiable formulas in constraint normal form by using that $<_2$ is a linear order (this leads to the restricted use of $<_2$-atoms in the constraints).

Lemma 5.6. *Every* $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$*-formula can be effectively transformed into an equisatisfiable formula in constraint normal form.*

For the construction of $\mathcal{A}$ we will henceforth assume that we are given a $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$-formula in constraint normal form. Let Θ be the signature of φ and Σ the set of unary, quantifier-free types over Θ.

The automaton $\mathcal{A}$ works on input trees over Σ 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 $\mathcal{A}$ will be constructed such that it accepts a data tree over Σ if and only if φ 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 φ. We next outline this correspondence and then show how to construct $\mathcal{A}$.

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

On the other hand, a data tree t with distinct data values corresponds to the structure $\mathcal{S}$ whose domain 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* φ if its corresponding structure $\mathcal{S}$ satisfies φ.

Lemma 5.7. *From an* $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$*-formula* φ *in constraint normal form one can effectively construct a TRASI* $\mathcal{A}$ *such that*

(1) *for data trees t with distinct values, $\mathcal{A}$ accepts t if and only if t satisfies φ, and*
(2) *if $\mathcal{A}$ accepts some data tree, then it also accepts one with distinct data values.*

Thus the TRASI from the lemma ensures that a data tree is accepted if and only if φ has a model. Combining this lemma with the decision procedure for emptiness of TRASIs yields Proposition 5.5.

The TRASI we construct for proving Lemma 5.7 has to handle $\mathrm{MSO}(<_1)$ formulas occuring in constraints. To do so, it will store type-information of nodes. Recall that a k-ary MSO-type $\tau(x_1, \ldots, x_k)$ of quantifier-rank q is a maximal consistent subset of MSO-formulas of quantifier-rank q and free variables $x_1, \ldots, x_k$ (see [Lib04, Section 7.2] for background on MSO-types). For a tree t with ancestor order $<$ and an $\mathrm{MSO}(<)$-type τ, we write $(t, u_1, \ldots, u_k) \models \tau$ if the tuple $(u_1, \ldots, u_k)$ of nodes in t is of type τ.

It is folklore that the rank-q MSO-type of a tree can be constructed from the rank-q MSO-types of its components. This is implicit in Shelah's Theorem on *generalised sums* [She75], see also the exposition from Blumensath et al. [BCL08, Theorem 3.16].

We will use the following special cases.

Lemma 5.8. *Suppose v is a node of a tree t. If u is a node in $t \setminus t_v$, then the rank-q MSO-type of (u, v) can be effectively obtained from the rank-q MSO-types of (1) $(\uparrow v, u)$ in $t \setminus t_v$, (2) $\swarrow v$ in $t_{\swarrow v}$, (3) $\searrow v$ in $t_{\searrow v}$, and of v. Similarly if u is a node of $t_{\swarrow v}$ or $t_{\searrow v}$, or if $u = v$.*

Lemma 5.9. *For every binary $\mathrm{MSO}(<)$-type $\tau(x, y)$ 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 with ancestor order $<$ there is a labelling with symbols from Γ which is $\mathrm{MSO}(<)$-definable and such that for all pairs (u, v) of nodes of t with $u < v$: $(t, u, v) \models \tau$ if and only if the labels on the path from u to v constitute a string from L.*

We are now ready to proceed with the proof of Lemma 5.7.

Proof of Lemma 5.7. The automaton $\mathcal{A}$ is the intersection of two automata $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neq}$. The automaton $\mathcal{A}_\varphi$ verifies that an input tree satisfies φ 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 $\mathcal{A}_\varphi$ (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.

We first explain the construction of the automaton $\mathcal{A}_\varphi$. This automaton checks whether an input tree t satisfies each of the existential and universal constraints in φ. To this end, it will (1) guess and verify $\mathrm{MSO}(<_1)$-type information for each node (recall that $<_1$ corresponds to the tree order), in order to handle the $\mathrm{MSO}(<_1)$-formulas η_i, ϑ_i and η in the constraints, and (2) use additional registers for checking the constraints themselves.

The automaton $\mathcal{A}_\varphi$ 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 $\mathrm{MSO}(<_1)$-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 $\mathcal{A}_\varphi$ 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 φ; 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($<_1$)-type $\tau(x,y)$ of quantifier-rank at most q. At a node v of an input tree t, these registers are intended to store the supremum and infimum data value over all nodes u with $(t,v,u) \models \tau$, that is it is intended that

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

for register assignments 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 $(v,u) \models \tau$. This can be tested via a constraint $r_{\tau,\sup}(v) > d(v)$. The verification of universal constraints is analogous.

The values of the registers in (A) are verified with registers from a set (B) of registers. The intuition for registers in (B) is as follows. For a node v, the supremum of data values of nodes u with $(t,v,u) \models \tau$ 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:

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

If $(t,v,v) \models \tau$, 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 $(t \setminus t_v, \uparrow v, u) \models \gamma$,

(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 $(t_{\swarrow v}, \swarrow v, u) \models \gamma$,

(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 $(t_{\searrow v}, \searrow v, u) \models \gamma$,

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 following equation:

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

Here, composition of types is according to Lemma 5.8.

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 additional registers $r_{\tau,\sup,\downarrow}$ with an associated language L, such that the supremum of these registers with respect to L-descendants is the supremum over all nodes u with $(t_v, v, u) \models \tau$.

Thus the values of $r_{\tau,\sup,\swarrow}$ and $r_{\tau,\sup,\searrow}$ can be verified by comparing them to these supremuma. The register $r_{\tau,\sup,\downarrow}$ stores the input data value of a node v. Its associated language L is the language over alphabet Γ for τ according to Lemma 5.9. 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 following equation:

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

This equation can be implemented easily by a tree register automaton with additional auxiliary registers. This concludes the construction of $\mathcal{A}_\varphi$.

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 $\mathcal{A}_\varphi$, 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 $t \setminus t_v$.

We show how a TRASI $\mathcal{A}_{\neq}$ can check that the data values of an input data tree for $\mathcal{A}_\varphi$ are weakly diverse. For this it has to be verified that no two nodes v and v' with the same data value d are connected by a path on which each node has a register with value d. In other words, no value d of a register may originate from the input register s of two different nodes v and v'.

Towards checking this condition, $\mathcal{A}_{\neq}$ guesses a *source* (v, r) for each data value d, where v is a node and r is a register. It ensures that on all nodes the input register is a source. Now, verifying that no data value has two sources ensures that the assignment is weakly diverse. For this verification, sources are annotated by labels from $\{S_r \mid r \text{ is a register}\}$ with the intention that a node u is labelled by S_r if (u, r) is a source. For tracing values to their source, each node is annotated by labels from $\{S_r^\uparrow, S_r^\swarrow, S_r^\searrow \mid r \text{ is a register}\}$ with the intention that, e.g., if a node u is labelled by $S_r^\swarrow$ then the value of register r at node u originates somewhere in the left subtree of u.

The automaton then checks the following for each node u:

(1) The node u is labelled by exactly one of the labels $S_r, S_r^\uparrow, S_r^\swarrow$, or $S_r^\searrow$, for each register r. Furthermore the labels are consistent for registers with the same value, i.e. if $\alpha(u, r) = \alpha(u, r')$ then u is labelled by S_r if and only if it is labeled by $S_{r'}$, and similarly for $S_r^\uparrow$, $S_r^\swarrow$, and $S_r^\searrow$.

(2) The labels S_r, $S_r^\uparrow, S_r^\swarrow$, or $S_r^\searrow$ indicate the direction of the source of the data value of register r at node u:
  (a) If u is the left child of its parent v and $\alpha(u, r) = \alpha(v, r')$, then

- If u is labelled by S_r, $S_r^{\searrow}$ or $S_r^{\nwarrow}$ then v is labelled by $S_{r'}^{\nwarrow}$.
- If u is labelled by $S_r^{\uparrow}$ then v is labelled by $S_{r'}$, $S_{r'}^{\searrow}$, or $S_{r'}^{\uparrow}$.

(b) Similarly if u is the right child of v, and if u is the parent of v.

(3) The register s of node u is a source, i.e. each node u is labelled by S_s. (Recall that s is a distinguished input register.)

The first condition ensures that no data value originates from register s of two different nodes. The automaton $\mathcal{A}_{\neq}$ now guesses and verifies the described annotation witnessing weak diversity for the input register s with respect to the registers of $\mathcal{A}_{\varphi}$. $\qquad\square$

5.1.2. *All Tree Orders interpret in the Complete Binary Tree.* A tree order is *universal* if it contains every countable tree order as a substructure. Denote by $\mathcal{T}$ the complete binary tree encoded as a structure over $\{L, R, <\}$ which interprets L and R as the left and right child relations, and $<$ as the ancestor order of the complete binary tree.

We strongly suspect that the following result is known; for lack of reference we provide a proof sketch.

Lemma 5.10. *There are* MSO-*formulas $\delta_U(x)$ and $\delta_{\prec}(x, y)$ over signature $\{L, R, <\}$ such that the structure $(U, \prec)$ with*

$$U \overset{df}{=} \{a \in V \mid \mathcal{T} \models \varphi(a)\}$$

$$\prec \overset{df}{=} \{(a, b) \in U \times U \mid \mathcal{T} \models \varphi(a, b)\}$$

is a universal tree order.

Before proving the lemma, we outline how it implies Theorem 5.1.

Proof of Theorem 5.1. Suppose φ is an $\mathrm{EMSO}^2(\mathrm{MSO}(<_1), <_2)$ formula for a tree order $<_1$ and a linear order $<_2$. Then satisfiability of φ reduces to the $\mathrm{EMSO}^2(\mathrm{MSO}(<_1'), <_2)$ formula $\varphi' = \exists S \psi$ where $<_1'$ is the ancestor order of the full binary tree and ψ is obtained from φ by

(1) relativizing all first-order quantifiers via replacing subformulas $\exists x \gamma(x)$ and $\forall x \gamma(x)$ by $\exists x (S(x) \wedge \delta_{U(x)} \wedge \gamma(x))$ and $\forall x (S(x) \wedge \delta_{U(x)} \to \gamma(x))$, respectively, and

(2) replacing all atoms $x <_1 y$ by $\delta_{\prec}(x, y)$.

Here, $\delta_{U(x)}$ and $\delta_{\prec}(x, y)$ are the formulas from Lemma 5.10.

Thus, a tree order $<_1$ in a model of φ is simulated in models of φ' by guessing a substructure (of domain S) of the universal tree order defined by $\delta_{U(x)}$ and $\delta_{\prec}(x, y)$. $\qquad\square$

We proceed with the proof of Lemma 5.10. The following notations and facts will be used in the proof. For a tree order $\prec$, we say that x is an *ancestor* of y, or that y is a *descendant* of x, if $x \prec y$ or $x = y$, written $x \preccurlyeq y$. The *meet* $x \wedge y$ of x and y is the greatest lower bound of x and y, with respect to $\preccurlyeq$. Observe that if $x, y_1, \ldots, y_m$ are elements of the tree order, then $x \wedge y_1, \ldots, x \wedge y_m$ are ancestors of x (if they exist). In particular, due to the definition of tree orders, the tree order $\prec$ is total on $x \wedge y_1, \ldots, x \wedge y_m$.

Proof sketch. Instead of working with universality directly, we use the following stronger condition. A nonempty tree order $\prec$ satisfies the *extension axioms* if

(1) for any node x and nodes $y_1, \ldots, y_n$ which are mutually incomparable descendants of x, there exists yet another incomparable descendant y,

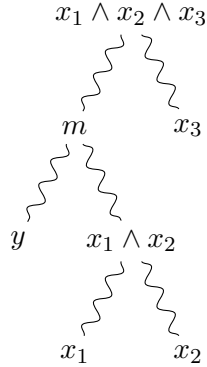(2) for every node x there is some y such that $y \prec x$, and

FIGURE 5. *Illustration of the construction of the mapping from V' to V in the proof of Lemma 5.10. Suppose the elements x_1, x_2, x_3 are already in X. Due to the closure under meets, also $x_1 \wedge x_2$ and $x_1 \wedge x_2 \wedge x_3$ are in X. The set M contains the nodes $m = y \wedge x_2$ and nodes $y \wedge x_3$.*

(3) for any two nodes x and y such that $x \prec y$, there is a node z such that $x \prec z \prec y$.

Let us first show that every tree order $\prec$ on a set of nodes V satisfying the extension axioms is universal. Let $\prec'$ be another tree order with countable domain V'. Extending this tree order by countably many points if necessary, we may assume that $\prec'$ is closed under meets, i.e., the meet of any two elements of V' is in V'. Fix an enumeration of the elements of the domain V'.

We inductively construct an embedding $f : V' \to V$ by defining f for larger and larger finite meet-closed subsets X of V'. For the induction basis, the first element of V' with respect to the chosen enumeration is mapped to an arbitrary element of V. Now suppose that we have already embedded a finite subset X of V' into V. In the inductive step we extend the embedding by considering the first node y of $V' - X$ with respect to the chosen enumeration. Consider the set

$$M \stackrel{df}{=} \{y \wedge x \mid x \in X\} \subseteq V'$$

of meets of y with the previously inserted nodes (see Figure 5). This set is a finite set of ancestors of y, hence it has a largest element m with respect to $\prec'$. We observe that the set $M \setminus \{m\}$ is contained in X due to the meet closure of X.

If the element m does not already belong to X then we first extend f to $X \cup \{m\}$. To this end let b be the image under f of the $\prec'$-smallest element of the set $\{x \in X \mid x \succ' m\}$, which exists since X is meet-closed.

If $|M| = 1$, apply the second extension axiom to b in order to obtain an element $a \in V$ with $a \prec b$, and define $f(m) \stackrel{df}{=} a$.

If $|M| > 1$ then let b' be the image under f of the second-largest element of M (the second largest element must be in X, due to meet-closure). Apply the third extension axiom to b' and b in order to obtain an element c of V' with $b' \prec c \prec b$ to which m will now be mapped.

Once f is defined on $X \cup \{m\}$, use the first extension axiom to extend the embedding to y. Finally, replace X by $X \cup \{m, y\}$ and proceed to the next element of $V' - X$. Note that the set $X \cup \{m, y\}$ is closed under meet.

This concludes the proof that $\prec$ is universal.

We will now show that a tree order satisfying the extension axioms can be first-order interpreted in the complete binary tree labelled with an additional unary relation U that can be axiomatized by a first-order formula (there is a formula $\phi(U)$ which holds exactly in one expansion of the complete binary tree by a unary predicates U).

Step 1. The complete ranked ternary tree (each node has exactly one "left child", one "right child" and one "middle child") and its tree order can be first-order interpreted into the complete binary tree using additional unary relations. This is immediate, by using a unary predicate $U \subseteq \{0,1\}^*$ satisfying the following properties:
- the root ε of the tree belongs to U,
- if v belongs to U then $v00, v01, v1 \in U$ and $v0 \notin U$.

Note that U is unique and the above conditions can be expressed by a first-order formula using the predicate U.

Step 2. The $\mathbb{Q}$-*tree* is the rooted tree where each non-root node has a parent, and the children of each node are ordered "from left to right" forming a dense order without endpoints. The $\mathbb{Q}$-tree, its tree order, and the order on siblings interpret in the complete ranked ternary tree T.

As a warm-up we observe that the structure $(\mathbb{Q}, \leq)$ interprets in the complete ranked ternary tree T. The elements of $(\mathbb{Q}, \leq)$ correspond to the nodes of T which are middle children, and which have no ancestor which is a middle child. The order $\leq$ is given by $v \leq w$ if and only if v is the left descendant of the meet of v and w.

For interpreting the $\mathbb{Q}$-tree in the complete ranked ternary tree, the construction for $(\mathbb{Q}, \leq)$ is nested. More explicitly, the nodes of the $\mathbb{Q}$-tree are the nodes of T which are middle children, and the ancestor relation is the ancestor relation of T restricted to those nodes. The left-to-right order on the children v, w (in the $\mathbb{Q}$-tree) of a given node is such that v is to the left of w if and only if v is the left descendant of the meet of v and w in T.

Step 3. Consider the interpretation of a tree order $\prec$ in the $\mathbb{Q}$-tree, in which $v \prec w$ if and only if v is a left-sibling of an ancestor of w in the $\mathbb{Q}$-tree. It is easy to see that $\prec$ is indeed a tree order. Also all meets exist in $\prec$, since, for all elements x and y, there is a meet z of x and y in the $\mathbb{Q}$-tree with children x' and y' which are ancestors of x and y. The smaller of the two nodes x' and y' with respect to the children order of the $\mathbb{Q}$-tree is the meet of x and y.

We further claim that $\prec$ satisfies the extension axioms, apart from the second one. For showing the first extension axiom, suppose that $y_1, \ldots, y_n$ are pairwise incomparable with respect to $\prec$ and such that $x \prec y_i$ for all i. Since $x \prec y_i$ for all i there must be a node with children $x_1, \ldots, x_n$ such that each x_i is an ancestor of y_i in the $\mathbb{Q}$-tree, and without loss of generality the x_i appear in the order $x_1, \ldots, x_n$. Since siblings in the $\mathbb{Q}$-tree are dense, there is a sibling x' to the left of x_1 such that $x \prec x'$. Every child y of x in the $\mathbb{Q}$-tree is incomparable to $y_1, \ldots, y_n$ with respect to $\prec$.

Towards showing that $\prec$ satisfies the third extension axiom, suppose that $x \prec y$. If x is not a left-sibling of y then choose some left-sibling z of y, otherwise choose an arbitrary z in-between x and y in the $\mathbb{Q}$-tree (such a z exists as siblings in the $\mathbb{Q}$-tree are dense). In both cases $x \prec z \prec y$.

The order $\prec$ does not satisfy the second extension axiom, as it contains a smallest element, corresponding to the root of the $\mathbb{Q}$-tree. However, it follows from the third extension

axiom that after removing the root, the second extension axiom is satisfied. Therefore, $\prec$ restricted to the set of non-root nodes of the $\mathbb{Q}$-tree is a universal tree order.

It is not hard to verify that all steps are indeed definable in MSO. $\qquad\square$

5.2. **Satisfiability of** $\mathrm{ESO}^2(<_1, <_2)$ **reduces to** $\mathrm{EMSO}^2(<_1, <_2)$. 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 $\mathrm{EMSO}^2(\mathcal{K})$. Examples are two-variable logic with one linear order [Ott01] and two-variable logic with one relation to be interpreted by a preorder [Pra18].

Here we show that the same approach works for ordered structures with two linear orders. This proves Corollary 5.4.

Proposition 5.11. *The satisfiability problem of* $\mathrm{ESO}^2(<_1, <_2)$ *effectively reduces to the satisfiability problem of* $\mathrm{EMSO}^2(<_1, <_2)$, *for linear orders* $<_1$ *and* $<_2$.

For the proof of this proposition, we rely on a general reduction for a broad range of classes of structures which is implicit in the procedure used by Pratt-Hartmann in [Pra18]. We make this approach explicit and show that it can also be applied to 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. One might wonder, why not concentrate on FO^2 throughout, yet we prefer to state results such as Proposition 5.11 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.

The high level proof idea is as follows. Recall that a FO^2-formula can be transformed in a satisfiability-preserving way into Scott normal form. For structures with at least two elements, formulas in Scott normal form can be transformed into an equisatisfiable formula in *standard normal form* [Pra18], that is, into the form

$$\forall x \forall y (x = y \vee \psi(x, y)) \wedge \bigwedge_{h=1}^{m} \forall x \exists y (x \neq y \wedge \psi_h(x, y))$$

where both ψ and all ψ_h are quantifier-free. Thus, a formula in standard normal form poses some universal as well as some existential constraints on models. Each existential constraint $\forall x \exists y (x \neq y \wedge \psi_h(x, y))$ requires that all instantiations a of x come with a *witness* instantiation b of y such that a and b satisfy ψ_h.

We will show that (1) under certain conditions, the $\forall\exists$-constraints can be required to *spread* their witnesses over disjoint elements (see Lemma 5.13), and (2) if witnesses are spread in this way, then binary relation symbols can be removed from formulas in a satisfiability-preserving way (see Lemma 5.14). Proposition 5.11 will then follow from the observation that $\mathrm{FO}^2(<_1, <_2)$ allows to spread witnesses (see Lemma 5.12).

We start by refining the standard normal form to spread normal form in which witnesses of existential constraints $\forall x \exists y (x \neq y \wedge \psi_h(x, y))$ are spread over disjoint elements. More precisely, our goal is that if such a refined formula has a model, then witnesses for $\forall x \exists y (x \neq y \wedge \psi_h(x, y))$ for an element a are 'spread', that is, (i) the witnesses for $\psi_0, \ldots, \psi_{m-1}$ are distinct elements, and (ii) if a has a witness b then all witnesses for b are not the element a. This intuition is

formalized in the following notion. Fix a class $\mathcal{K}$ of structures over some signature Δ. An $\mathrm{FO}^2(\mathcal{K})$ formula is in *spread normal form* [Pra18, p. 23] if it conforms to the pattern

$$\bigwedge_{\theta \in Z} \exists x \theta(x) \wedge \forall x \forall y (x = y \vee \nu(x, y))$$

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

where (i) Z is a set of unary pure Boolean formulas; (ii) $\nu, \theta_0, \ldots, \theta_{m-1}$ are quantifier- and equality-free formulas; and (iii) $\lambda_0, \lambda_1,$ 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 intended and stated by Pratt-Hartmann, "the essential feature of formulas in spread normal form is that witnesses are 'spread' over disjoint sets of elements": (i) the witnesses for the individual m existential conjuncts are distinct elements – ensured by the mutually exclusive μ_h; and (ii) if a has a witness b then all witnesses for b are not the element a – ensured by the mutually exclusive λ_k.

Our next goal is to find a general criterion for when $\mathrm{FO}^2(\mathcal{K})$ formulas can be transformed into equisatisfiable formulas in spread normal form, for a given class $\mathcal{K}$ of structures of signature Δ. Intuitively, $\mathrm{FO}^2(\mathcal{K})$ formulas φ can be transformed into a equisatisfiable formula φ^* in spread normal form, if elements can be cloned without affecting the truth of the formula. Indeed, if elements can be cloned, then disjoint witnesses for existential constraints can be created on-demand by cloning elements. In a moment, after formalizing what it means that an element can be cloned, we will see that not all elements can be cloned. This will lead us to slightly adapt our goal and to rather look for a criterion for when $\mathrm{FO}^2(\mathcal{K})$ formulas can be transformed into a *disjunction* of equisatisfiable formulas in spread normal form, where each disjunct essentially takes care of one configuration of the non-clonable elements.

Let us formalize what it means that elements can be cloned. We first fix some notation. In the following we denote the quantifier-free type of a tuple $\bar{a}$ in a structure $\mathcal{A}$ by $\mathrm{TP}^{\mathcal{A}}[\bar{a}]$. We say that a quantifier-free type τ is *realized* in $\mathcal{A}$, if there is a tuple $\bar{a}$ with $\mathrm{TP}^{\mathcal{A}}[\bar{a}] = \tau$. A structure $\mathcal{A}$ over signature $\Delta \cup \Theta$ is an expansion of a Δ-structure $\mathcal{B}$, if $\mathcal{A}$ and $\mathcal{B}$ have the same domain and agree on the interpretations of relation symbols from Δ.

Now, 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 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) $\mathcal{A}$ is a substructure of $\mathcal{A}'$ 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.

It is easy to see that it is not possible to clone elements whose unary quantifier-free type is unique within a structure: when we clone such an element b, then the binary type of b and its clone b' has not been present before, which violates condition (i) from above. This motivates the following definition. An element a of a structure $\mathcal{A}$ is a *King*, if there
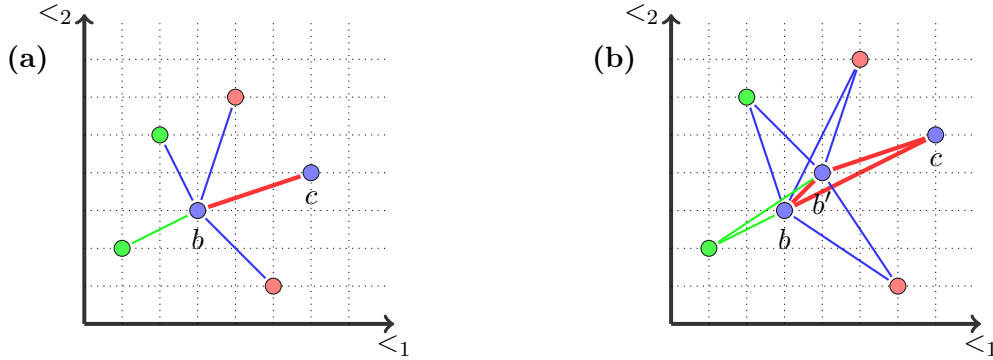
FIGURE 6. **(a)** *A structure with non-King elements b and c with the same unary type.* **(b)** *The structure obtained by cloning the element b once. For clarity only the binary types between b and its clone b' with respect to other elements are indicated.*

is no other element b in $\mathcal{A}$ with the same quantifier-free type, i.e., there is no $b \neq a$ with $\mathrm{TP}^{\mathcal{A}}[a] = \mathrm{TP}^{\mathcal{A}}[b]$.

Now, we say that a *class $\mathcal{K}$ of structures allows cloning* if every expansion of a $\mathcal{K}$-structure allows cloning of arbitrary sets of non-King elements. The following lemma establishes that in structures with two linear orders we can clone arbitrary sets of non-King elements. The same construction can be used for structures with more than two linear orders.

Lemma 5.12. *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 $A' \overset{df}{=} A \cup \{b' \mid b \in B\}$. The relations between elements in A within the structure $\mathcal{A}'$ are inherited from $\mathcal{A}$. The simple idea for new elements b' 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'. See Figure 6 for an illustration.

More formally, the tuples of $\mathcal{A}'$ have the following types :

(i) $\mathrm{TP}^{\mathcal{A}'}[a] = \mathrm{TP}^{\mathcal{A}}[a]$ and $\mathrm{TP}^{\mathcal{A}'}[a, a'] = \mathrm{TP}^{\mathcal{A}}[a, a']$ for all $a, a' \in A$;

(ii) $\mathrm{TP}^{\mathcal{A}'}[b'] = \mathrm{TP}^{\mathcal{A}}[b]$ and $\mathrm{TP}^{\mathcal{A}'}[b', a] = \mathrm{TP}^{\mathcal{A}}[b, a]$ for all $b \in B$ and $a \in A$;

(iii) $\mathrm{TP}^{\mathcal{A}'}[b_1', b_2'] = \mathrm{TP}^{\mathcal{A}'}[b_1, b_2]$ for all $b_1, b_2 \in B$ with $b_1 \neq b_2$;

(iv) $\mathrm{TP}^{\mathcal{A}'}[b, b'] = \mathrm{TP}^{\mathcal{A}}[b, c]$ for $c \in A$ with $\mathrm{TP}^{\mathcal{A}}(c) = \mathrm{TP}^{\mathcal{A}}(b)$.

It can be easily verified that $\mathcal{A}'$ satisfies the conditions on cloning. $\qquad \square$

When a class $\mathcal{K}$ allows cloning then witnesses for a formula $\varphi \in \mathrm{FO}^2(\mathcal{K})$ in Scott normal form can be spread among cloned elements. Unfortunately, Kings need to be dealt with separately as they cannot be cloned. But, essentially, for each possible "configuration" of Kings and their witnesses one can find a formula in spread normal form such that their disjunction is equisatisfiable to φ.

Lemma 5.13 (Adaption of Lemma 4.2 in [Pra18]). *If a class $\mathcal{K}$ of structures allows cloning then for all $\varphi \in \mathrm{FO}^2(\mathcal{K})$ one can effectively find formulas $(\varphi_\mathcal{C})_{\mathcal{C} \in C}$, for some finite set C, in spread normal form such that the following conditions are equivalent:*

(1) *φ is (finitely) satisfiable.*

(2) *$\bigvee_{\mathcal{C} \in C} \varphi_\mathcal{C}$ is (finitely) satisfiable.*

Proof. This follows immediately from the proof of Lemma 4.2 in [Pra18]. The structures $\mathcal{A}_1, \ldots, \mathcal{A}_i$ used in that proof can be easily constructed inductively by successively cloning non-King elements. The rest of the proof is independent of the class $\mathcal{K}$.

For the sake of completeness we provide a proof sketch. Suppose $\mathcal{K}$ contains structures over signture Δ and φ is a $\mathrm{FO}^2(\mathcal{K})$ formula over signature $\Delta \uplus \Theta$ in standard normal form:

$$\forall x \forall y (x = y \lor \psi(x, y)) \land \bigwedge_{h=1}^{m} \forall x \exists y (x \neq y \land \psi_h(x, y))$$

Each of the formulas $\varphi_\mathcal{C}$ in spread normal form that we will construct is responsible for one configuration of Kings and their witnesses in models of φ. More precisely, let Γ_K be a set of unary types over the signature of φ. A *court structure for Γ_K and φ* is a structure $\mathcal{C}$ such that

(1) the types of Kings in $\mathcal{C}$ are exactly the types in Γ_K,

(2) $\mathcal{C}$ satisfies $\forall x \forall y (x = y \lor \psi(x, y))$, and

(3) for all Kings a of $\mathcal{C}$ and all $h \in \{1, \ldots, m\}$ there is exactly one b in $\mathcal{C}$ such that $C \models \psi_h(a, b)$.

Thus, such a court structure represents a set of Kings as well as their witnesses. Denote by C the set of all court structures for all sets Γ_K of unary types.

For each $\mathcal{A}$ with $\mathcal{A} \models \varphi$, the *court structure induced by $\mathcal{A}$* is the substructure of $\mathcal{A}$ induced by the Kings of $\mathcal{A}$ as well as their witnesses.

Our goal is to construct formulas $\varphi_\mathcal{C}$ in spread normal form for each $\mathcal{C} \in C$ such that the following conditions are equivalent:

(1) φ has a (finite) model with court structure $\mathcal{C}$.

(2) $\varphi_\mathcal{C}$ has a (finite) model.

Then the statement of the proposition follows by observing that φ has a (finite) model if and only if $\bigvee_{\mathcal{C} \in C}$ has a (finite) model.

Thus we aim at constructing, for each $\mathcal{C} \in C$, a formula $\varphi_\mathcal{C}$ in spread normal form with the above property. Recall that a formula is in spread normal form, if it is of the form

$$\bigwedge_{\theta \in Z} \exists x \theta(x) \land \forall x \forall y (x = y \lor \nu(x, y))$$

$$\land \bigwedge_{k=0}^{2} \bigwedge_{h=1}^{n} \forall x \exists y (\lambda_k(x) \to (\lambda_{\lfloor k+1 \rfloor}(y) \land \mu_h(y) \land \theta_h(x, y)))$$

where (i) Z is a set of unary pure Boolean formulas; (ii) $\nu, \theta_1, \ldots, \theta_n$ are quantifier- and equality-free formulas; and (iii) $\lambda_0, \lambda_1,$ and λ_2 are mutually exclusive unary pure Boolean formulas; and (iv) $\mu_1, \ldots, \mu_n$ are mutually exclusive unary pure Boolean formulas.

The formula $\varphi_\mathcal{C}$ ensures a set of conditions on models. Suppose that $\mathcal{C}$ has the elements $a_1, \ldots, a_t$ ordered in such a way that $a_1, \ldots, a_s$ are its Kings and $a_{s+1}, \ldots, a_t$ are its non-King elements.

A first set of conditions, encoded in formulas δ_1 and δ_2, ensures that the court structure of models of $\varphi_{\mathcal{C}}$ are isomorphic to $\mathcal{C}$. To this end, let $Q_1, \ldots, Q_t$ be fresh unary relation symbols.

The formula δ_1 ensures that each such relation Q_i contains exactly one element (intended to be the ith element of $\mathcal{C}$):

$$\bigwedge_{i=1}^{t} \exists x Q_i(x) \ \wedge \ \forall x \forall y \Big(x = y \vee \bigwedge_{i=1}^{t} \neg(Q_i(x) \wedge Q_i(y)) \Big)$$

The formula δ_2 ensures that the substructure induced by elements in the Q_is is isomorphic to $\mathcal{C}$:

$$\bigwedge_{i=1}^{t-1} \bigwedge_{j=i+1}^{t} \forall x \forall y \Big(x = y \vee ((Q_i(x) \wedge Q_j(y)) \rightarrow \text{TP}^{\mathcal{C}}[a_i, a_j]) \Big)$$

A second set of conditions, encoded in formulas γ_1, γ_2, and γ_3 ensures that witnesses are spread. To this end, let $Q_{h,1}, \ldots, Q_{h,s}$ be fresh unary relation symbols. The intuition is that $Q_{h,i}$ shall contain all elements a such that the ψ_h-witness of a is the ith King. For spreading non-King witnesses, further fresh unary relation symbols $W_1, \ldots, W_m$ and N_0, N_1, N_2 are used.

Witnesses for an element a will then be ensured by distinguishing between King witnesses, handled by γ_2, and non-King witnesses, handeled by γ_3.

The formula γ_1 enforces that each element is either a King, or occurs in exactly one W_i and in exactly one N_i:

$$\forall x \forall y \Big(x = y \vee \bigvee_{i=1}^{s} Q_i(x) \vee \mu(x) \Big)$$

Here, $\mu(x)$ is a formula stating that x occurs in exactly one W_i and in exactly one N_i. (Note that the quantification of y is not really necessary, but makes it easy to see that γ_2 adheres to the conditions for spread normal form.)

The formula γ_2 ensures that King witnesses are present:

$$\bigwedge_{i=1}^{s} \bigwedge_{h=1}^{m} \forall x \forall y \Big(x = y \vee ((Q_{h,i}(x) \wedge Q_i(y)) \rightarrow \psi_h(x, y)) \Big)$$

Finally, the formula γ_3 ensures that each element has its non-King witnesses:

$$\bigwedge_{k=0}^{2} \bigwedge_{h=1}^{m} \forall x \exists y \Big(N_k(x) \rightarrow (N_{\lfloor k+1 \rfloor}(y) \wedge W_h(y) \wedge \psi_h^*(x, y)) \Big)$$

where $\psi_h^*(x, y)$ is of the form

$$\psi_h^*(x, y) \stackrel{df}{=} \Big(\bigwedge_{i=1}^{s} \neg Q_{h,i}(x) \Big) \rightarrow \psi_h(x, y)$$

Now, formula $\varphi_{\mathcal{C}}$ can be obtained by transforming the conjunction of δ_1, δ_2, γ_1, γ_2, γ_3, and $\nu \stackrel{df}{=} \forall x \forall y (x = y \vee \psi(x, y))$ into spread normal form. Due to the form of these formulas, this can be easily achieved. This concludes the construction of $\varphi_{\mathcal{C}}$

It remains to show that φ has a (finite) model with court structure $\mathcal{C}$ if and only if $\varphi_{\mathcal{C}}$ has a (finite) model.

From a model $\mathcal{A}_{\mathcal{C}}$ of $\varphi_{\mathcal{C}}$ one can construct a model $\mathcal{A}$ of φ by ignoring the interpretations of all new relation symbols in $\varphi_{\mathcal{C}}$, that is, by ignoring the relations $Q_i^{\mathcal{A}_{\mathcal{C}}}$, $Q_{h,i}^{\mathcal{A}_{\mathcal{C}}}$, $W_i^{\mathcal{A}_{\mathcal{C}}}$ and $N_i^{\mathcal{A}_{\mathcal{C}}}$. The thus obtained structure $\mathcal{A}$ satisfies the $\forall\forall$-constraints of φ due to ν, and the $\forall\exists$-constraints due to δ_2, γ_2, and γ_3. The court structure induced by $\mathcal{A}$ is $\mathcal{C}$ due to δ_2.

Conversely suppose that $\mathcal{A}$ is a model of φ with induced court structure $\mathcal{C}$. A model $\mathcal{A}_{\mathcal{C}}$ of $\varphi_{\mathcal{C}}$ can be constructed from $\mathcal{A}$ as follows. Let A be the universe of $\mathcal{A}$, let $A_0 \stackrel{df}{=} \{a_1, \ldots, a_s\}$ be its set of Kings, and let $B \stackrel{df}{=} A \setminus A_0$.

The structure $\mathcal{A}_{\mathcal{C}}$ is constructed by cloning B. It has universe $A_0 \cup \bigcup_{k=0}^{2} \bigcup_{h=1}^{m} B_{k,h}$ where each $B_{k,h}$ is a copy of B.

The relations $Q_i^{\mathcal{A}_{\mathcal{C}}}$ and $Q_{h,i}^{\mathcal{A}_{\mathcal{C}}}$ are defined as:

$$Q_i^{\mathcal{A}_{\mathcal{C}}} \stackrel{df}{=} \{a_i\} \quad \text{and} \quad Q_{h,i}^{\mathcal{A}_{\mathcal{C}}} \stackrel{df}{=} \{b' \mid b' \text{ is a copy of an element } b \text{ with } \psi_h(b, a_i)\}$$

The relations $W_h^{\mathcal{A}_{\mathcal{C}}}$ and $N_k^{\mathcal{A}_{\mathcal{C}}}$ are defined as:

$$W_h^{\mathcal{A}_{\mathcal{C}}} \stackrel{df}{=} \bigcup_{k=0}^{2} B_{k,h} \quad \text{and} \quad N_k^{\mathcal{A}_{\mathcal{C}}} \stackrel{df}{=} \bigcup_{h=1}^{m} B_{k,h}$$

It remains to argue that the thus constructed structure $\mathcal{A}_{\mathcal{C}}$ is a model of $\varphi_{\mathcal{C}}$. The formulas δ_1 and δ_2 are satisfied, as $\mathcal{C}$ is the court structure induced by $\mathcal{A}$. The formula γ_1 is satisfied due to the construction of $W_h^{\mathcal{A}_{\mathcal{C}}}$ and $N_k^{\mathcal{A}_{\mathcal{C}}}$. The formula γ_2 is satisfied due the construction of $Q_{h,i}^{\mathcal{A}_{\mathcal{C}}}$. The formula γ_3 is satisfied due to the construction of $W_h^{\mathcal{A}_{\mathcal{C}}}$ and $N_k^{\mathcal{A}_{\mathcal{C}}}$, and because the $\forall\exists$-constraints of φ are satisfied in $\mathcal{A}$. For seeing that ν is satisfed we observe that it is satisfied in $\mathcal{A}$, and cloning does not lead to new binary types (see (i) in the definition of cloning). $\qquad\square$

The proofs of Lemma 4.3, Lemma 4.4, and Theorem 4.5 in [Pra18] implicitly imply the following lemma.

Lemma 5.14. *Suppose $\mathcal{K}$ is a class of structures with signature Δ and φ is a $\mathrm{FO}^2(\mathcal{K})$ formula over $\Delta \uplus \Theta$. If φ is in spread normal form then one can effectively construct an equisatisfiable $\mathrm{FO}^2(\mathcal{K})$ formula over signature $\Delta \uplus \Theta'$ where Θ' only contains unary relation symbols.*

Proof. We provide two ways of proving the statement.

Adaption of the proof of Pratt-Hartmann [Pra18]. The proofs of Lemma 4.3 and Lemma 4.4 in [Pra18] are independent of the class $\mathcal{K}$ of structures. Thus, in particular, Lemma 4.4 can be restated for arbitrary classes $\mathcal{K}$ with a spread normal form.

Suppose that φ is an $\mathrm{FO}^2(\mathcal{K})$ formula in spread normal form with signature $\Delta \uplus \Theta$. By Lemma 4.4 in [Pra18], φ can be translated into φ° such that (1) φ° does not use atoms of the form $r(x, y)$ or $r(y, x)$ for $r \in \Theta$, (2) the formula φ implies φ°, and (3) if φ° has a model over domain A, then so has φ. As the only binary atoms from Θ occurring in φ° are of the form $r(x, x)$ or $r(y, y)$, they can be replaced by new unary atoms, see Theorem 4.5 in [Pra18]. This yields the intended equisatisfiable formula.

Direct proof. Let φ be a $\mathrm{FO}^2(\mathcal{K})$ formula over signature $\Delta \uplus \Theta$ in spread normal form

$$\bigwedge_{\theta \in Z} \exists x \theta(x) \wedge \forall x \forall y (x = y \vee \nu(x, y))$$

$$\wedge \bigwedge_{k=0}^{2} \bigwedge_{h=0}^{m-1} \forall x \exists y (\lambda_k(x) \to (\lambda_{\lfloor k+1 \rfloor}(y) \wedge \mu_h(y) \wedge \theta_h(x, y)))$$

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.

Suppose, without loss of generality, that the formulas ν and θ_h are of the form

$$\bigvee_i \alpha_i(x, y)$$

where all α_i are binary types.

For a binary type α, denote by α^* the set of formulas obtained from α by removing all literals of the form $R(x, y)$, $R(y, x)$, $\neg R(x, y)$, and $\neg R(y, x)$ for $R \in \Theta$. Further denote by ν^* and θ_h^* the formulas obtained by replacing all binary types α by α^* in ν and θ_h.

Consider the formula φ^* defined as

$$\bigwedge_{\theta \in Z} \exists x \theta(x) \wedge \forall x \forall y (x = y \vee \nu^*(x, y))$$

$$\wedge \bigwedge_{k=0}^{2} \bigwedge_{h=0}^{m-1} \forall x \exists y (\lambda_k(x) \to (\lambda_{\lfloor k+1 \rfloor}(y) \wedge \mu_h(y) \wedge \theta_h^*(x, y)))$$

This formula only uses literals from Δ, unary literals over Θ and binary literals of the form $R(x, x), R(y, y), \neg R(x, x), \neg R(y, y)$ over Θ.

We claim that φ^* and φ are equisatisfiable. Certainly a model $\mathcal{A}$ of φ is also a model of φ^* (as can be seen by inspecting the conjuncts of φ^*).

For the converse suppose that $\mathcal{A}^*$ is a model of φ^*. Then a model $\mathcal{A}$ of φ can be obtained as follows (cf. proof of Lemma 4.4 in [Pra18]). The structure $\mathcal{A}$ has the same elements as $\mathcal{A}^*$. Further $\mathcal{A}$ inherits the unary types from $\mathcal{A}^*$. It remains to fix the binary types for $\mathcal{A}$.

Fix an element a. If a satisfies one of the λ_i (and since these are disjoint, no other λ_j), then for all h there exists b such that $\mathcal{A} \models \lambda_{i+1}(b) \wedge \mu_h(b)$. Let α^* be a disjunct of θ_h^* that is satisfied by (a, b). Suppose α^* was obtained from a type β of θ_h in the construction of φ^*, then chose β as the binary type of (a, b). Note that β is consistent with the (already assigned) unary types of a and b, as α^* was obtained from β by removing strictly binary literals. In this way, all witnesses for a are fixed. Since all μ_h are disjoint (since φ is in spread normal form), all such witnesses are distinct.

The above procedure is repeated for all a. Note that for a tuple (a, b) the binary types assigned for determining the witnesses of a and b do not conflict since witnesses satisfy $\lambda_{\lfloor i+1 \rfloor}$ which is disjoint from λ_i. Furthermore, the types assigned in this fashion do not conflict with ν, because all types α used in the formulas θ_h must be consistent with ν.

It remains to assign the binary types in $\mathcal{A}$ for all remaining tuples (a, b). This has to be done in accordance with the condition ν. As $\mathcal{A}^*$ satisfies φ^*, the tuple (a, b) satisfies ν^* and hence some formula α^* that was obtained from some binary type β in ν. The binary type of (a, b) is chosen as β. This, again, does not conflict with previously chosen unary types of a and b.

Hence φ and φ^* are equisatisfiable. Now, an equisatisfiable formula ψ for φ over signature $\Delta \uplus \Theta'$ where Θ' only contains unary relation symbols can be obtained from φ^* as follows (cf. proof of Theorem 4.5 in [Pra18]). All binary atoms in φ^* with relation symbols from Θ are of the form $R(x,x)$ or $R(y,y)$. Replacing these atoms by atoms $\widehat{R}(x)$ and $\widehat{R}(y)$ where $\widehat{R}$ is a fresh unary relation symbol yields the equisatisfiable formula ψ over signature $\Delta \uplus \Theta'$ where

$$\Theta' \overset{df}{=} \{S \in \Theta \mid S \text{ is unary}\} \uplus \{\widehat{R} \in \Theta \mid R \in \Theta \text{ is binary}\}.$$

$\square$

Proof (of Proposition 5.11). By Lemma 5.12 structures with two linear order allow cloning, and therefore each $\text{FO}^2(<_1, <_2)$ formula can be transformed into an equisatisfiable disjunction $\bigvee_i \varphi_i$ of formulas φ_i over signature $\{<_1, <_2\} \uplus \Theta_i$ in spread normal form by Lemma 5.13. Now, each φ_i can be transformed into an equisatisfiable formula φ_i' over $\{<_1, <_2\} \uplus \Theta_i'$ where each Θ_i' is unary, by Lemma 5.14. Now, Proposition 5.11 follows from the fact that $\bigvee_i \varphi_i'$ is equisatisfiable to $\bigvee_i \varphi_i$. $\square$

Acknowledgment

References

[BCL08] Achim Blumensath, Thomas Colcombet, and Christof Löding. Logical theories and compatible operations. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, pages 73–106, 2008.

[BKL08] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.

[BMSS09] Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):13:1–13:48, 2009.

[CW13] Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and trees. In *LICS'13*, pages 73–82, 2013.

[DFL18] Luc Dartois, Emmanuel Filiot, and Nathan Lhote. Logics for word transductions with synthesis. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 295–304. ACM, 2018.

[GKV97] Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

[GO99] Erich Grädel and Martin Otto. On logics with two variables. *Theor. Comput. Sci.*, 224(1-2):73–113, 1999.

[JLZ20] Jean Christoph Jung, Carsten Lutz, and Thomas Zeume. On the decidability of expressive description logics with transitive closure and regular role expressions. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 529–538, 2020.

[KF94] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

[Kie11] Emanuel Kieronski. Decidability issues for two-variable logics with several linear orders. In *CSL 2011*, volume 12 of *LIPIcs*, pages 337–351, 2011.

[KO12]   Emanuel Kieronski and Martin Otto. Small substructures and decidability issues for first-order logic with two variables. *J. Symb. Log.*, 77(3):729–765, 2012.

[KT09]   Emanuel Kieronski and Lidia Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *LICS 2009*, pages 123–132. IEEE Computer Society, 2009.

[Lib04]   Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[Mor75]   Michael Mortimer. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 21:135–140, 1975.

[Nev02]   Frank Neven. Automata theory for XML researchers. *ACM Sigmod Record*, 31(3):39–46, 2002.

[NSV04]   Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

[Ott01]   Martin Otto. Two variable first-order logic over ordered domains. *J. Symb. Log.*, 66(2):685–702, 2001.

[Pra18]   Ian Pratt-Hartmann. Finite satisfiability for two-variable, first-order logic with one transitive relation is decidable. *Math. Log. Q.*, 64(3):218–248, 2018.

[Rab72]   Michael Oser Rabin. *Automata on Infinite Objects and Church's Problem*. Number 13. American Mathematical Soc., 1972.

[Sco62]   Dana Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27(377):74, 1962.

[She75]   Saharon Shelah. The monadic theory of order. *Annals of Mathematics*, pages 379–419, 1975.

[SZ12]   Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. *Logical Methods in Computer Science*, 8(1), 2012.

[Tho97]   Wolfgang Thomas. Languages, automata, and logic. In *Handbook of formal languages*, pages 389–455. Springer, 1997.

[ZH16]   Thomas Zeume and Frederik Harwath. Order-invariance of two-variable logic is decidable. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 807–816. ACM, 2016.