Extending Two-Variable Logic on Data Trees with Order on Data Values and Its Automata

TONY TAN, Hasselt University and Transnational University of Limburg

Data trees are trees in which each node, besides carrying a label from a finite alphabet, also carries a data value from an infinite domain. They have been used as an abstraction model for reasoning tasks on XML and verification. However, most existing approaches consider the case where only equality test can be performed on the data values.

In this article we study data trees in which the data values come from a linearly ordered domain, and in addition to equality test, we can test whether the data value in a node is greater than the one in another node. We introduce an automata model for them which we call *ordered-data tree automata* (ODTA), provide its logical characterisation, and prove that its non-emptiness problem is decidable in 3-NExpTime. We also show that the two-variable logic on unranked data trees, studied by Bojanczyk et al. [2009], corresponds precisely to a special subclass of this automata model.

Then we define a slightly weaker version of ODTA, which we call *weak ODTA*, and provide its logical characterisation. The complexity of the non-emptiness problem drops to NP. However, a number of existing formalisms and models studied in the literature can be captured already by weak ODTA. We also show that the definition of ODTA can be easily modified, to the case where the data values come from a tree-like partially ordered domain, such as strings.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Computational logic*

General Terms: Languages

Additional Key Words and Phrases: Finite-state automata, two-variable logic, data trees, ordered data values

ACM Reference Format:
Tony Tan. 2014. Extending two-variable logic on data trees with order on data values and its automata. ACM Trans. Comput. Logic 15, 1, Article 8 (February 2014), 39 pages.
DOI: http://dx.doi.org/10.1145/2559945

1. INTRODUCTION

Classical automata theory studies words and trees over finite alphabets. Recently there has been a growing interest in the so-called data words and trees, that is, words and trees in which each position, besides carrying a label from a finite alphabet, also carries a data value from an infinite domain.

Interest in such structures with data springs due to their connection to XML [Alon et al. 2003; Arenas et al. 2008; Björklund et al. 2008; David et al. 2012; Fan and Libkin 2002; Figueira 2009; Neven 2002], as well as system specifications [Bouyer et al. 2001; Demri et al. 2007; Segoufin and Torunczyk 2011], where many properties simply cannot be captured by finite alphabets. This has motivated various works on data

T. Tan was suppported under the FWO Pegasus Marie Curie fellowship.
The extended abstract of this article appears in *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS'12)*.

DOI: http://dx.doi.org/10.1145/2559945

words [Benedikt et al. 2010; Bojanczyk et al. 2011a; Demri and Lazić 2009; Grumberg et al. 2010; Kaminski and Francez 1994; Neven et al. 2004], as well as on data trees [Björklund and Bojanczyk 2007; Bojanczyk et al. 2009; Figueira 2012a; Figueira and Segoufin 2011; Jurdzinski and Lazic 2011]. The common feature of these works is the addition of equality test on the data values to the logic on trees. While for finitely-labeled trees, many logical formalisms (e.g., the monadic second-order logic MSO) are decidable by converting formulae to automata, even FO (first-order logic) on data words extended with data-equality is already undecidable see, e.g., Bojanczyk et al. [2011a], Fan and Libkin [2002], and Neven et al. [2004].

Thus, there is a need for expressive enough, while computationally well-behaved, frameworks to reason about structures with data values. This has been quite a common theme in XML and system specification research. It has largely followed two routes. The first takes a specific reasoning task, or a set of similar tasks, and builds algorithms for them (see, e.g., Arenas et al. [2008], Figueira [2011], Björklund et al. [2008], Schwentick [2004], Fan and Libkin [2002], and Figueira [2009]). The second looks for sufficiently general automata models that can express reasoning tasks of interest, but are still decidable (see, e.g., Demri and Lazić [2009], Bojanczyk et al. [2009], Jurdzinski and Lazic [2011], and Segoufin and Torunczyk [2011]).

Both approaches usually assume that data values come from an abstract set equipped only with the equality predicate. This is already sufficient to capture a wide range of interesting applications both in databases and verification. However, it has been advocated in Deutsch et al. [2009] that comparisons based on a linear order over the data values could be useful in many scenarios, including data-centric applications built on top of a database.

So far, not many works have been done in this direction. A few works such as Manuel [2010], Figueira [2011], Schwentick and Zeume [2010], and Segoufin and Torunczyk [2011] are on words, while in most applications, we need to consider trees. Moreover, these works are incomparable to some interesting existing formalisms [Fan and Libkin 2002; Bojanczyk et al. 2009; Arenas et al. 2008; David et al. 2012; Jurdzinski and Lazic 2011; Demri and Lazić 2009; Lazić 2011] known to be able to capture various interesting scenarios common in practice. On top of that, many useful techniques, notably those introduced in Fan and Libkin [2002], Bojanczyk et al. [2011a], Bojanczyk et al. [2009], and Jurdzinski and Lazic [2011], can deal only with data equality, and are highly dependent on specific combinatorial properties of the formalisms. They are rather hard to adapt to other more specific tasks, let alone being generalised to include more relations on data values, and they tend to produce extremely high complexity bounds, such as non-primitive-recursive, or at least as hard as the reachability problem in Petri nets. Furthermore, many known decidability results are lost as soon as we add the order relation on data values. Some exceptions are Figueira et al. [2010, 2012a].

In this article, we study the notion of data trees in which the data values come from a linearly ordered domain, which we call *ordered-data trees*. In addition to equality tests on the data values, in ordered-data trees, we are allowed to test whether the data value in a node is greater than the data value in another node. To the extent it is possible, we aim to unify various ad hoc methods introduced to reason about data trees, and generalise them to ordered-data trees to make them more accessible and applicable in practice. This article is the first step, where we introduce an automata model for ordered-data trees, provide its logical characterisation, and prove that it has decidable non-emptiness problem. Moreover, we also show that it can capture various well known formalisms.

Brief Description of the Results in this Article. The trees, that we consider are *unranked* trees where there is no a priori bound in the number of children of a node.

Moreover, we also have an order on the children of each node. We consider a natural logic for ordered-data trees, which consists of the following relations.

—The parent relation $E_{\downarrow}$, where $E_{\downarrow}(x, y)$ means that node x is the parent of node y.
—The next-sibling relation $E_{\rightarrow}$, where $E_{\rightarrow}(x, y)$ means that nodes x and y have the same parent and y is the next sibling of x.
—The labeling predicates $a(\cdot)$'s, where $a(x)$ means that node x is labeled with symbol a.
—The data equality predicate $\sim$, where $x \sim y$ means that nodes x and y have the same data value.
—The order relation on data $\prec$, where $x \prec y$ means that the data value in node x is less than the one in node y.
—The successive order relation on data $\prec_{suc}$, where $x \prec_{suc} y$ means that the data value in node y is the minimal data value in the tree greater than the one in node x.

We introduce an automata model for ordered-data trees, which we call *ordered-data tree automata* (ODTA), and provide its logical characterisation. Namely, we prove that the class of languages accepted by ODTA corresponds precisely to those expressible by formulas of the form:

$$\exists X_1 \cdots \exists X_n \, \varphi \wedge \psi, \tag{1}$$

where

—$X_1, \ldots, X_n$ are monadic second-order predicates;
—φ is an FO formula restricted to two variables and using only the predicates $E_{\downarrow}$, $E_{\rightarrow}$, $\sim$, as well as the unary predicates $X_1, \ldots, X_n$ and a's;
—ψ is an FO formula using only the predicates $\sim$, $\prec$, $\prec_{suc}$, as well as the unary predicates $X_1, \ldots, X_n$ and a's.

We show that the logic $\exists\text{MSO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$, first studied in Bojanczyk et al. [2009], corresponds precisely to a special subclass of ODTA, where $\exists\text{MSO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$ denotes the set of formulas of the form of Eq. (1) in which ψ is a true formula. We then prove that the non-emptiness problem of ODTA is decidable in 3-NExpTime. Our main idea here is to show how to convert the ordered-data trees back to a string over *finite* alphabets (see our notion of *string representation of data values* in Section 3). Such conversion enables us to use the classical finite state automata to reason about data values.

Then we define a slightly weaker version of ODTA, which we call *weak ODTA*. Essentially, the only feature of ODTA missing in weak ODTA is the ability to test whether two adjacent nodes have the same data value. Without such simple feature, the complexity of the non-emptiness problem surprisingly drops three-fold exponentially to NP. We provide its logical characterisation by showing that it corresponds precisely to the languages expressible by the formulas of the form of Eq. (1), where φ does not use the predicate $\sim$. We show that a number of existing formalisms and models can be captured already by weak ODTA, that is, those in Fan and Libkin [2002], David et al. [2012], and Manuel [2010].

We should remark that David et al. [2012] studies a formalism which consists of tree automata and a collection of *set* and *linear* constraints.[1] It is shown that the satisfiability problem of such formalism is NP-complete. In fact, it is also shown [David et al. 2012] that a single set constraint (without tree automaton and linear constraint) already yields NP-hardness. Weak ODTA are essentially equivalent to the formalism in David et al. [2012] extended with the full expressive power of the first-order logic

[1]We will later define formally what set and linear constraints are.

FO($\sim, \prec, \prec_{suc}$). It is worth to note that despite such extension, the non-emptiness problem remains in NP.

Finally, we also show that the definition of ODTA can be easily modified to the case where the data values come from a partially ordered domain, such as strings. This work can be seen as a generalisation of the works in David et al. [2010] and Kara et al. [2012]. However, it must be noted that David et al. [2010] and Kara et al. [2012] deal only with *data words*, where only equality test is allowed on the data values and there is no order on them.

Related Works. Most of the existing works in this area are on data words. Bojanczyk et al. [2011a] introduce the model *data automata*, and it was shown that it captures the logic $\exists\text{MSO}^2(\sim, <, +1)$, the fragment of existential monadic second-order logic in which the first-order part uses only two variables and the predicates: the data equality $\sim$, as well as the order $<$ and the successor $+1$ on the domain.

An important feature of data automata is that their non-emptiness problem is decidable, even for infinite words, but is at least as hard as reachability for Petri nets. It was also shown that the satisfiability problem for the three-variable first-order logic is undecidable. Later, in David et al. [2010], an alternative proof was given for the decidability of the weaker logic $\exists\text{MSO}^2(+1, \sim)$. The proof gives a decision procedure with an elementary upper bound for the satisfiability problem of $\exists\text{MSO}^2(+1, \sim)$ on strings. Recently, in Kara et al. [2012], an automata model that captures precisely the logic $\exists\text{MSO}^2(+1, \sim)$, both on finite and infinite words, is proposed.

Another logical approach is via the so called *linear temporal logic* with freeze quantifier, introduced in Demri and Lazić [2009]. Intuitively, these are LTL formulas equipped with a finite number of registers to store the data values. We denote by $\text{LTL}^{\downarrow}_n[\mathtt{X}, \mathtt{U}]$, the LTL with freeze quantifier, where n denotes the number of registers and the only temporal operators allowed are the neXt operator $\mathtt{X}$ and the Until operator $\mathtt{U}$. It was shown that alternating register automata with n registers (RA_n) accept all $\text{LTL}^{\downarrow}_n[\mathtt{X}, \mathtt{U}]$ language, and the non-emptiness problem for alternating RA_1 is decidable. However, the complexity is nonprimitive recursive. Hence, the satisfiability problem for $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$ is decidable as well. Adding one more register or past time operator $\mathtt{U}^{-1}$ to $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$ makes the satisfiability problem undecidable. Figueira et al. [2010, Figueira 2012a], it is shown that alternating RA_1 can be extended to strings with linearly ordered data values, and the emptiness problem is still decidable. In Lazić [2011], a weaker version of alternating RA_1, called safety alternating RA_1, is considered, and the non-emptiness problem is shown to be EXPSPACE-complete.

A model for data words with linearly ordered data values was proposed in Segoufin and Torunczyk [2011]. The model consists of an automaton equipped with a finite number of registers, and its transitions are based on constraints on the data values stored in the registers. It is shown that the non-emptiness problem for this model is decidable in PSPACE. However, no logical characterisation is provided for such a model.

In Bojanczyk et al. [2011b] another type of register automata for words was introduced and studied, which is a generalisation of the original register automata introduced by Kaminski and Francez [1994], where the data values also can come from a linearly ordered domain. Thus, the order comparison, not just equality, can be performed on data values. The generalisation is done via the notion of monoid for data words and is incomparable with our model here. In the terminology of the original register automata defined in Kaminski and Francez [1994], it is simply register automata extended with testing whether the data value currently read is bigger/smaller than those in the registers.

It is shown in Manuel [2010] that the satisfiability problem for $\mathrm{FO}^2(+1, \prec_{suc})$ over *text* is decidable. A *text* is simply a data word in which all the data values are different and they range over the positive integers from 1 to n, for some $n \geq 1$. We will see later that the satisfiability problem for $\mathrm{FO}^2(+1, \prec_{suc})$ can be reduced to the non-emptiness problem of our model.

In Schwentick and Zeume [2010], it is shown that the satisfiability problem of the logic $\mathrm{FO}^2(<, \prec)$ on *words* is decidable. This logic is incomparable with our model. However, it should be noted that $\mathrm{FO}^2(<)$ cannot capture the whole class of regular languages.

The work on data trees that we are aware of is in Bojanczyk et al. [2009] and Jurdzinski and Lazic [2011]. In Bojanczyk et al. [2009], it was shown that the satisfiability problem for the logic $\exists\mathrm{MSO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$ over unranked trees is decidable in 3-NExpTime. However, no automata model is provided. We will see later how this logic corresponds precisely to a special subclass of ODTA.

In Jurdzinski and Lazic [2011], alternating tree register automata were introduced for trees. They are essentially the generalisation of the alternating RA_1 to the tree case. It was shown that this model captures the forward XPath queries. However, no logical characterisation is provided, and the nonemptiness problem, though decidable, is nonprimitive recursive.

As mentioned earlier, the main idea in this article is the conversion of the data values from an infinite domain back to string over a finite alphabet. Roughly speaking, it works as follows. Given an ordered-data tree t, we show how to construct a string w over a finite alphabet whose domain corresponds precisely to the data values in t. We then use the classical finite state automaton to reason about w, and thus, also about the data values in t. This idea is the main difference between our article and the existing works. Most of the existing techniques rely on some specific combinatorial properties of the formalisms considered, which make them highly independent of one another. As we will see later, our model captures quite a few other formalisms without significant jump in complexity.

Organisation. This article is organised as follows. In Section 2, we give some preliminary background. In Section 3, we formally define the logic for ordered-data trees and present a few examples as well as notations that we need in this article. In Section 4, we present two lemmas that we are going to need later on. We prove them in a quite general setting, as we think they are interesting in their own. We introduce the ordered-data tree automata (ODTA) in Section 5 and weak ODTA in Section 6. In Section 7, we discuss a couple of the undecidable extensions of weak ODTA. In Section 8, we describe how to modify the definition of ODTA when the data values are strings, that is, when they come from a partially ordered domain. Finally we conclude with some concluding remarks in Section 9.

2. PRELIMINARIES

In this section, we review some definitions that we are going to use later on. We usually use Γ and Σ to denote finite alphabets. We write 2^{Γ} to denote an alphabet in which each symbol corresponds to a subset of Γ. In some cases, we may need the alphabet $2^{2^{\Gamma}}$—an alphabet in which each symbol corresponds to a set of subsets of Γ. We denote the set of natural numbers $\{0, 1, 2, \ldots\}$ by $\mathbb{N}$.

Usually we write $\mathcal{L}$ to denote a language, for both string and tree languages. When it is clear from the context, we use the term *language* to mean either a string language, or a tree language.

2.1. Finite-State Automata over Strings and Commutative Regular Languages

We usually write $\mathcal{M}$ to denote a finite-state automaton on strings. The language accepted by the automaton $\mathcal{M}$ is denoted by $\mathcal{L}(\mathcal{M})$.

Let $\Sigma = \{a_1, \ldots, a_\ell\}$. For a word $w \in \Sigma^*$, the Parikh image of w is $\mathsf{Parikh}(w) = (n_1, \ldots, n_\ell)$, where n_i is the number of appearances of a_i in w. For a vector $\bar{n}$, the inverse of the Parikh image of $\bar{n}$ is $\mathsf{Parikh}^{-1}(\bar{n}) = \{w \mid w \in \Sigma^* \text{ and } \mathsf{Parikh}(w) = \bar{n}\}$.

For $1 \leq i \leq \ell$, a vector $\bar{v} = (n_1, \ldots, n_\ell) \in \mathbb{N}^\ell$ is called an *i-base*, if $n_i \neq 0$ and $n_j = 0$, for all $j \neq i$. A language $\mathcal{L}$ is *periodic*, if there exist $(\ell + 1)$ vectors $\bar{u}, \bar{v}_1, \ldots, \bar{v}_\ell$ such that $\bar{u} \in \mathbb{N}^\ell$ and each $\bar{v}_i$ is an i-base and

$$\mathcal{L} = \bigcup_{h_1, \ldots, h_\ell \geq 0} \mathsf{Parikh}^{-1}(\bar{u} + h_1\bar{v}_1 + \cdots + h_\ell\bar{v}_\ell).$$

We denote such language $\mathcal{L}$ by $\mathcal{L}(\bar{u}, \bar{v}_1, \ldots, \bar{v}_\ell)$.

A language $\mathcal{L}$ is *commutative* if it is closed under reordering, that is, if $w = b_1 \cdots b_m \in \mathcal{L}$, and σ is a permutation on $\{1, \ldots, m\}$, then $b_{\sigma(1)} \cdots b_{\sigma(m)} \in \mathcal{L}$.

The following result is a kind of folklore and can be proved easily.

THEOREM 2.1. *A language is commutative and regular if and only if it is a finite union of periodic languages.*

2.2. Unranked Trees, Tree Automata and Transducers

An unranked finite tree domain is a prefix-closed finite subset D of $\mathbb{N}^*$ (words over $\mathbb{N}$) such that $u \cdot i \in D$ implies $u \cdot j \in D$ for all $j < i$ and $u \in \mathbb{N}^*$. Given a finite labeling alphabet Σ, a Σ-labeled unranked tree t is a structure

$$\langle D, E_\downarrow, E_\rightarrow, \{a(\cdot)\}_{a \in \Sigma} \rangle,$$

where

—D is an unranked tree domain,
—$E_\downarrow$ is the child relation: $(u, u \cdot i) \in E_\downarrow$ for all $u, u \cdot i \in D$,
—$E_\rightarrow$ is the next-sibling relation: $(u \cdot i, u \cdot (i + 1)) \in E_\rightarrow$ for all $u \cdot i, u \cdot (i + 1) \in D$, and
—the $a(\cdot)$'s are labeling predicates, that is, for each node u, exactly one of $a(u)$, with $a \in \Sigma$, is true.

We write $\mathsf{Dom}(t)$ to denote the domain D. The label of a node u in t is denoted by $\ell ab_t(u)$. If $\ell ab_t(u) = a$, then we say that u is an a-node.

An *unranked tree automaton* [Comon et al. 2007; Thatcher 1967] over Σ-labeled trees is a tuple $\mathcal{A} = \langle Q, \Sigma, \delta, F \rangle$, where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \to 2^{(Q^*)}$ is a transition function; we require $\delta(q, a)$'s to be regular languages over Q for all $q \in Q$ and $a \in \Sigma$.

A run of $\mathcal{A}$ over a tree t is a function $\rho_\mathcal{A} : \mathsf{Dom}(t) \to Q$ such that for each node u with n children $u \cdot 0, \ldots, u \cdot (n-1)$, the word $\rho_\mathcal{A}(u \cdot 0) \cdots \rho_\mathcal{A}(u \cdot (n-1))$ is in the language $\delta(\rho_\mathcal{A}(u), \ell ab_t(u))$. For a leaf u labeled a, this means that u could be assigned a state q if and only if the empty word ϵ is in $\delta(q, a)$. A run is accepting if $\rho_\mathcal{A}(\epsilon) \in F$, that is, if the root is assigned a final state. A tree t is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on t. The set of all trees accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$.

An unranked tree (letter-to-letter) transducer with the input alphabet Σ and output alphabet Γ is a tuple $\mathcal{T} = \langle \mathcal{A}, \mu \rangle$, where $\mathcal{A}$ is a tree automaton with the set of states Q, and $\mu \subseteq Q \times \Sigma \times \Gamma$ is an output relation. We call such $\mathcal{T}$ a transducer from Σ to Γ.

Let t be a Σ-labeled tree, and t' a Γ-labeled tree such that $\mathsf{Dom}(t) = \mathsf{Dom}(t')$. We say that a tree t' is an output of $\mathcal{T}$ on t, if there is an accepting run $\rho_\mathcal{A}$ of $\mathcal{A}$ on t and for each $u \in \mathsf{Dom}(t)$, it holds that $(\rho_\mathcal{A}(u), \ell ab_t(u), \ell ab_{t'}(u)) \in \mu$. We call $\mathcal{T}$ an identity transducer,

if $\ell ab_t(u) = \ell ab_{t'}(u)$ for all $u \in \mathsf{Dom}(t)$. We will often view an automaton $\mathcal{A}$ as an identity transducer.

2.3. Automata with Presburger Constraints (APC)

An automaton with Presburger constraints (APC) is a tuple $\langle \mathcal{A}, \xi \rangle$, where $\mathcal{A}$ is an unranked tree automaton with states $q_0, \ldots, q_m$ and ξ is an existential Presburger formula with free variables $x_0, \ldots, x_m$. A tree t is accepted by $\langle \mathcal{A}, \xi \rangle$, denoted by $t \in \mathcal{L}(\mathcal{A}, \xi)$, if there is an accepting run $\rho_{\mathcal{A}}$ of $\mathcal{A}$ on w such that $\xi(n_0, \ldots, n_m)$ is true, where n_i is the number of appearances of q_i in $\rho_{\mathcal{A}}$.

THEOREM 2.2 [SEIDL ET AL. 2004; VERMA ET AL. 2005]. *The non-emptiness problem for APC is decidable in* NP.

It is worth noting also that the class of languages accepted by APC is closed under union and intersection.

Oftentimes, instead of counting the number of states in the accepting run, we need to count the number of occurrences of alphabet symbols in the tree. Since we can easily embed the alphabet symbols inside the states, we always assume that the Presburger formula ξ has the free variables x_a's to denote the number of appearances of the symbol a in the tree.

As in the word case, we let $\mathsf{Parikh}(t)$ denote the Parikh image of the tree t. We will need the following proposition.

PROPOSITION 2.3 [SEIDL ET AL. 2004; VERMA ET AL. 2005]. *Given an unranked tree automaton $\mathcal{A}$, one can construct, in polynomial time, an existential Presburger formula $\xi_{\mathcal{A}}(x_1, \ldots, x_\ell)$ such that the following hold.*

—*For every tree $t \in \mathcal{L}(\mathcal{A})$, $\xi_{\mathcal{A}}(\mathsf{Parikh}(t))$ holds.*
—*For every $\bar{n} = (n_1, \ldots, n_\ell)$ such that $\xi_{\mathcal{A}}(\bar{n})$ holds, there exists a tree $t \in \mathcal{L}(\mathcal{A})$ with $\mathsf{Parikh}(t) = \bar{n}$.*

3. ORDERED-DATA TREES AND THEIR LOGIC

An ordered-data tree over the alphabet Σ is a tree in which each node, besides carrying a label from the finite alphabet Σ, also carries a data value from $\mathbb{N} = \{0, 1, \ldots\}$.[2]

Let t be an ordered-data tree over Σ and $u \in \mathsf{Dom}(t)$. We write $va\ell_t(u)$ to denote the data value in the node u. The set of all data values in the a-nodes in t is denoted by $V_t(a)$, that is, $V_t(a) = \{va\ell_t(u) \mid \ell ab_t(u) = a \text{ and } u \in \mathsf{Dom}(t)\}$. We write V_t to denote the set of data values found in the tree t. We also write $\#_t(a)$ to denote the number of a-nodes in t.

The profile of a node u is a triplet $(l, p, r) \in \{\top, \bot, *\} \times \{\top, \bot, *\} \times \{\top, \bot, *\}$, where $l = \top$ and $l = \bot$ indicate that the node u has the same data value and different data value as its left sibling, respectively; $l = *$ indicates that u does not have a left sibling. Similarly, $p = \top$, $p = \bot$, and $p = *$ have the same meaning in relation to the parent of the node u, while $r = \top$, $r = \bot$, and $r = *$ means the same in relation to the right sibling of the node u. For an ordered-data tree t over Σ, the profile tree of t, denoted by $\mathsf{Profile}(t)$, is a tree over $\Sigma \times \{\top, \bot, *\}^3$ obtained by augmenting to each node of t its profile.

We write $\mathsf{Proj}(t)$ to denote the Σ projection of the ordered-data tree t, that is, $\mathsf{Proj}(t)$ is t without the data values. When we say that an ordered-data tree t is accepted by an automaton $\mathcal{A}$, we mean that $\mathsf{Proj}(t)$ is accepted by $\mathcal{A}$. An ordered-data tree t' is an

[2]Here we use the natural numbers as data values just to be concrete. The results in our article apply trivially for any linearly ordered domain.

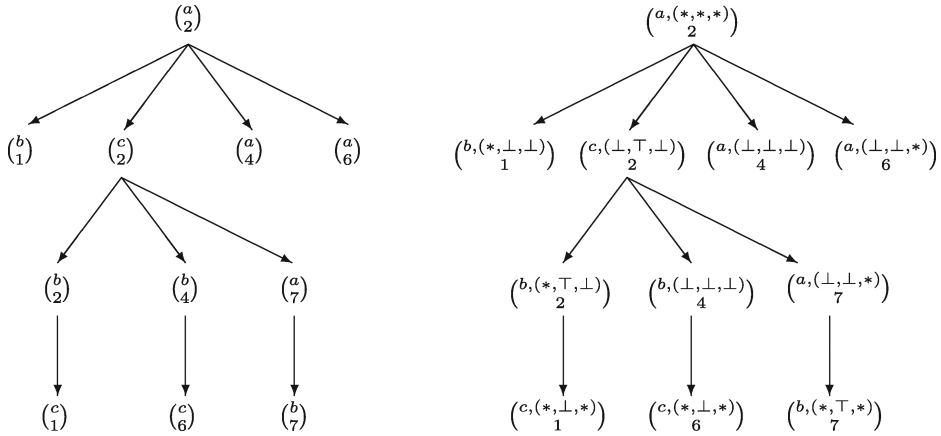

Fig. 1. An example of an ordered-data tree (on the left) and its profile (on the right).

output of a transducer $\mathcal{T}$ on an ordered-data tree t, if $\mathsf{Proj}(t')$ is an output of $\mathcal{T}$ on $\mathsf{Proj}(t)$, and for all $u \in \mathsf{Dom}(t')$, we have $va\ell_{t'}(u) = va\ell_t(u)$.

Figure 1 shows an example of an ordered-data tree t over the alphabet $\{a, b, c\}$ with its profile tree. The notation $\binom{a}{d}$ means that the node is labeled with a and has data value d.

3.1. String Representations of Data Values

Let t be an ordered-data tree over Γ. For a set $S \subseteq \Gamma$, let

$$[S]_t = \bigcap_{a \in S} V_t(a) \cap \bigcap_{b \notin S} \overline{V_t(b)}.$$

That is, $[S]_t$ is the set of data values that are found in a-positions for all $a \in S$ but are not found in any b-position for $b \notin S$. Note that the sets $[S]_t$'s are disjoint, and that for each $a \in \Gamma$,

$$V_t(a) = \bigcup_{S \text{ s.t. } a \in S} [S]_t.$$

Moreover, $|V_t(a)| = \sum_{S \text{ s.t. } a \in S} |[S]_t|$.

Let $d_1 < \cdots < d_m$ be all the data values found in t. The *string representation* of the data values in t, denoted by $\mathcal{V}_\Gamma(t)$, is the string $S_1 \cdots S_m$ over the alphabet $2^\Gamma - \{\emptyset\}$ of length m such that $d_i \in [S_i]_t$, for each $i = 1, \ldots, m$. The notation $[S]_t$ is already introduced in David et al. [2010, 2012], but not $\mathcal{V}_\Gamma(t)$.

Consider the example of the tree t in Figure 1. The data values in t are $1, 2, 4, 6, 7$, where

$$\begin{aligned} [\{b, c\}]_t &= \{1\}, \\ [\{a, b, c\}]_t &= \{2\}, \\ [\{a, b\}]_t &= \{4, 7\}, \\ [\{a, c\}]_t &= \{6\}, \\ [S]_t &= \emptyset, \text{ for all the other } S\text{'s}. \end{aligned}$$

The string $\mathcal{V}_\Gamma(t)$ is $S_1\, S_2\, S_3\, S_4\, S_5$, where $S_1 = \{b, c\}$, $S_2 = \{a, b, c\}$, $S_3 = S_5 = \{a, b\}$, and $S_4 = \{a, c\}$.

3.2. A Logic for Ordered-Data Trees

An ordered-data tree t over the alphabet Σ can be viewed as a structure

$$t = \langle D, \{a(\cdot)\}_{a\in\Sigma}, E_{\downarrow}, E_{\rightarrow}, \sim, \prec, \prec_{suc}\rangle,$$

where

—the relations $\{a(\cdot)\}_{a\in\Sigma}, E_{\downarrow}, E_{\rightarrow}$ are as defined in Section 2.2,
—$u \sim v$ holds, if $va\ell_t(u) = va\ell_t(v)$,
—$u \prec v$ holds, if $va\ell_t(u) < va\ell_t(v)$,
—$u \prec_{suc} v$ holds, if $va\ell_t(v)$ is the minimal data value in t greater than $va\ell_t(u)$.

Obviously, $x \prec_{suc} y$ can be expressed equivalently as $x \prec y \wedge \forall z(\neg(x \prec z \wedge z \prec y))$. We include $\prec_{suc}$ for the sake of convenience. We also assume that we have the predicates root(x), first-sibling(x), last-sibling(x), and leaf(x) which stand for $\forall y(\neg E_{\downarrow}(y, x))$, $\forall y(\neg E_{\rightarrow}(y, x))$, $\forall y(\neg E_{\rightarrow}(x, y))$, and $\forall y(\neg E_{\downarrow}(x, y))$, respectively. We also write $x \nsim y$ to denote $\neg(x \sim y)$.

For $\mathcal{O} \subseteq \{E_{\downarrow}, E_{\rightarrow}, \sim, \prec, \prec_{suc}\}$, we let FO($\mathcal{O}$) stand for the first-order logic with the vocabulary $\mathcal{O}$, MSO($\mathcal{O}$) for its monadic second-order logic (which extends FO($\mathcal{O}$) with quantification over sets of nodes), and $\exists$MSO($\mathcal{O}$) for its existential monadic second order logic, that is, formulas of the form $\exists X_1 \ldots \exists X_m\, \psi$, where ψ is an FO($\mathcal{O}$) formula over the vocabulary $\mathcal{O}$ extended with the unary predicates $X_1, \ldots, X_m$.

We let FO^2($\mathcal{O}$) stand for FO($\mathcal{O}$) with two variables, that is, the set of FO($\mathcal{O}$) formulae that only use two variables x and y. The set of all formulae of the form $\exists X_1 \ldots \exists X_m\, \psi$, where ψ is an FO^2($\mathcal{O}$) formula is denoted by $\exists$MSO^2($\mathcal{O}$). Note that $\exists$MSO^2($E_{\downarrow}, E_{\rightarrow}$) is equivalent in expressive power to MSO($E_{\downarrow}, E_{\rightarrow}$) over the usual (without data) trees. That is, it defines precisely the regular tree languages [Thomas 1997].

As usual, we define $\mathcal{L}_{data}(\varphi)$ as the set of ordered-data trees that satisfy the formula φ. In such case, we say that the formula φ expresses the language $\mathcal{L}_{data}(\varphi)$.[3]

The following theorem is well known. It shows how even extending FO($E_{\downarrow}, E_{\rightarrow}$) with equality test on data values immediately yields undecidability.

THEOREM 3.1 (E.G., [NEVEN ET AL. 2004]). *The satisfiability problem for the logic FO($E_{\downarrow}, E_{\rightarrow}, \sim$) is undecidable.*

One of the deepest results in this area is the following decidability result for the logic $\exists$MSO^2($E_{\downarrow}, E_{\rightarrow}, \sim$).

THEOREM 3.2 [BOJANCZYK ET AL. 2009]. *The satisfiability problem for the logic $\exists$MSO^2($E_{\downarrow}, E_{\rightarrow}, \sim$) is decidable.*

3.3. A Few Examples

In this section, we present a few examples of properties of ordered-data trees. Some of them are special cases of more general techniques that will be used later on.

Example 3.3. Let $\Sigma = \{a, b\}$. Consider the language $\mathcal{L}^a_{data}$ of ordered-data trees over Σ, where an ordered-data tree $t \in \mathcal{L}^a_{data}$ if and only if there exist two a-nodes u and v such that u is an ancestor of v and either $v \sim u$ or $v \prec u$. This language can be expressed with the formula $\exists X \exists Y \exists Z\, \varphi$, where φ states that X contains only the node u, Y contains only the node v, Z contains precisely the nodes in the path from u to v, and $v \sim u$ or $v \prec u$.

[3]To avoid confusion, we put the subscript *data* on $\mathcal{L}_{data}$ to denote a language of ordered-data trees. We use the symbol $\mathcal{L}$ without the subscript *data* to denote the usual language of trees/strings without data.

Example 3.4. For a fixed set $S \subseteq \Sigma$ and an integer $m \geq 1$, we consider the language $\mathcal{L}_{data}^{S,m}$ such that $t \in \mathcal{L}_{data}^{S,m}$ if and only if $|[S]_t| = m$.

We pick an arbitrary symbol $a \in S$. The language $\mathcal{L}_{data}^{S,m}$ can be expressed in $\exists\mathrm{MSO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$ with the formula of the form $\exists X_1 \cdots \exists X_m \varphi$, where φ is a conjunction of the following.

—That the predicates $X_1, \ldots, X_m$ are disjoint, and each of them contains exactly one node, which is an a-node.
—That the data values found in nodes in $X_1, \ldots, X_m$ are all different.
—That for each $i \in \{1, \ldots, m\}$, if a data value is found in a node in X_i, then it must also be found in some b-node, for every $b \in S$.
—That for each $i \in \{1, \ldots, m\}$, if a data value found in a node in X_i, then it must not be found in any b-node, for every $b \notin S$.
—That for every a-node (recall that $a \in S$) that does not belong to the X_i's, either it has the same data value as the data value in a node belongs to one of the X_i's, or it has the data value not in $[S]_t$.
That its data value does not belong to $[S]_t$ can be stated as the negation of the following.
—For each $b \in S$, there is a b-node with the same data value.
—The data value cannot be found in any b-node, for every $b \notin S$.

To express all these intended meanings, it is sufficient that $\varphi \in \mathrm{FO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$.

Example 3.5. For a fixed set $S \subseteq \Sigma$ and an integer $m \geq 1$, we consider the language $\mathcal{L}_{data}^{S,\ (\mathrm{mod}\ m)}$ such that $t \in \mathcal{L}_{data}^{S,\ (\mathrm{mod}\ m)}$ if and only if $|[S]_t| \equiv 0 \pmod{m}$.

This language $\mathcal{L}_{data}^{S,\ (\mathrm{mod}\ m)}$ can be expressed in $\exists\mathrm{MSO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$ with a formula of the form

$$\exists X_0 \cdots \exists X_{m-1} \exists Y_0 \cdots \exists Y_{m-1} \exists Z\ \psi,$$

where the intended meanings of $X_0, \ldots, X_{m-1}, Y_0, \ldots, Y_{m-1}, Z$ are as follows. For a node u in an ordered-data tree $t \in \mathcal{L}_{data}$, the following hold.

—The number of nodes belonging to Z is precisely $|[S]_t|$; and if $Z(u)$ holds in t, then the data value in the node u belongs to $[S]_t$.
—$X_i(u)$ holds in t if and only if in the subtree t' rooted in u we have $|[S]_{t'}| \equiv i \pmod{m}$.
—If $v_1, \ldots, v_k$ are all the left-siblings of u, and $X_{i_1}(v_1), \ldots, X_{i_k}(v_k)$ holds, then $Y_i(u)$ holds if and only if $i_1 + \cdots + i_k \equiv i \pmod{m}$.

To express all these intended meanings, it is sufficient that $\psi \in \mathrm{FO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$.

Example 3.6. Let $\Sigma = \{a, b\}$. Consider the language $\mathcal{L}_{data}^{a*}$ of ordered-data trees over Σ where an ordered-data tree $t \in \mathcal{L}_{data}^{a*}$ if and only if all the a-nodes with data values different from the ones in their parents satisfy the following conditions:

—the data values found in these nodes are all different;
—one of the these data values must be the largest in the tree t.

The language $\mathcal{L}_{data}^{a*}$ can be expressed in $\exists\mathrm{MSO}^2(E_{\downarrow}, \sim, \prec)$ with the following formula.

$$\begin{aligned}\exists X\ \Big(&\forall x\big(X(x) \iff a(x) \wedge \exists y(E_{\downarrow}(y,x) \wedge y \nsim x)\big)\\ &\wedge\ \forall x \forall y(X(x) \wedge X(y) \wedge x \sim y \rightarrow x = y)\\ &\wedge\ \exists x\big(X(x) \wedge \forall y(y \prec x \vee x \sim y)\big)\Big).\end{aligned}$$

4. TWO USEFUL LEMMAS

In this section, we prove two lemmas which will be used later on. The first is combinatorial by nature, and we will use it in our proof of the decidability of ODTA. The second is an Ehrenfeucht-Fraïssé type lemma for ordered-data trees, and we will use it in our proof of the logical characterization of ODTA.

4.1. A Combinatorial Lemma

Let G be an (undirected and finite) graph. For simplicity, we consider only the graph without self-loop. We denote by $V(G)$ the set of vertices in G and $E(G)$ the set of edges. For a node $u \in V(G)$, we write $\deg(u)$ to denote the degree of the node u and $\deg(G)$ to denote $\max\{\deg(u) \mid u \in V(G)\}$.

A *data graph* over the alphabet Γ is a graph G in which each node carries a label from Γ and a data value from $\mathbb{N}$. A node $u \in V(G)$ is called an a-node if its label is a, in which case we write $\ell ab_G(u) = a$. We denote by $va\ell_G(u)$ the data value found in node u, and $\mathrm{Val}_G(a)$ the set of data values found in a-nodes in G.

LEMMA 4.1. *Let G be a data graph over Γ. Suppose for each $a \in \Gamma$, we have $|V_G(a)| \geq \deg(G)|\Gamma| + \deg(G) + 1$. Then we can reassign the data values in the nodes in G to obtain another data graph G' such that $V(G) = V(G')$ and $E(G) = E(G')$ and*

(1) for each $u \in V(G')$, $\ell ab_G(u) = \ell ab_{G'}(u)$;
(2) for each $a \in \Gamma$, $Val_G(a) = Val_{G'}(a)$;
(3) for each $u, v \in V(G)$, if $(u, v) \in E(G')$, then $\mathrm{va}\ell_{G'}(u) \neq \mathrm{va}\ell_{G'}(v)$.

PROOF. Note that in the lemma, the data graph G' differs from G only in the data values on the nodes, where we require that adjacent nodes in G' have different data values.

In the following, we write $\#_G(a)$ to denote the number of a-nodes in G and $K = \deg(G)$. First, we perform some partial reassignment of the data values on some nodes. For each $a \in \Gamma$, we pick $|\mathrm{Val}_G(a)|$ number of a-nodes in G'. Then we assign to these a-nodes the data values from $\mathrm{Val}_G(a)$. One a-node gets one data value. Such assignment can be done, since obviously $\#_G(a) \geq |\mathrm{Val}_G(a)|$. If $\#_G(a) > |\mathrm{Val}_G(a)|$, then there will be some a-nodes in G' that do not have data values. We write $va\ell_{G'}(u) = \sharp$, if u does not have data value. From this step we already obtain that $\mathrm{Val}_{G'}(a) = \mathrm{Val}_G(a)$ for each $a \in \Gamma$.

However, reassigning the data values just like that, there may exist an edge (u, v) such that $va\ell_{G'}(u) = va\ell_{G'}(v)$ and $va\ell_{G'}(u), va\ell_{G'}(v) \neq \sharp$. We call such an edge a *conflict* edge. We are going to reassign the data values one more time so that there is no conflict edge.

Suppose there exists an edge $(u, v) \in E$ such that $va\ell_{G'}(u) = va\ell_{G'}(v) = d$, and suppose that u is an a-node, for some $a \in \Gamma$. The data value d can only be found in at most $|\Gamma|$ nodes in G'. Since $\deg(G) = K$, the neighbours of those nodes (with data value d) are at most $K|\Gamma|$ nodes. Now $|\mathrm{Val}_G(a)| = |\mathrm{Val}_{G'}(a)| \geq K|\Gamma| + K + 1$, there are at least $K + 1$ number of a-nodes whose neighbours do not get the data value d. Let $u_1, \ldots, u_m$ be such a-nodes, where $m \geq K + 1$. From these nodes, there exists i such that $va\ell_{G'}(u_i) \notin \{va\ell_{G'}(x) \mid (u, x) \in E\}$.

We can then swap the data values on the nodes u and u_i, and this results in one less conflict edge. We repeat this process until there is no conflict edge. Now it is straightforward that

(1) for each $u \in V$, $\ell ab_G(u) = \ell ab_{G'}(u)$;
(2) for each $a \in \Gamma$, $\mathrm{Val}_G(a) = \mathrm{Val}_{G'}(a)$;
(3) for each $u, v \in V$, if $(u, v) \in E$ and $va\ell_{G'}(u), va\ell_{G'}(v) \neq \sharp$, then $va\ell_{G'}(u) \neq va\ell_{G'}(v)$.

What is left to do now is to assign data values to the nodes u, where $va\ell_{G'}(u) = \sharp$. For each a-node, where $va\ell_{G'}(u) = \sharp$, we pick the data value $d \in \mathrm{Val}_{G'}(a) = \mathrm{Val}_G(a)$ which is not assigned to any its neighbour. Such data value exists, since $|\mathrm{Val}_{G'}(a)| \geq K|\Gamma| + K + 1 \geq K + 1$. Such assignment will not violate condition (3), thus, we get the desired data graph G'. This completes the proof of Lemma 4.1. □

4.2. An Ehrenfeucht-Fraïssé Type Lemma

We need the following notation. A k-characteristic function on the alphabet Γ is a function $f : \Gamma \to \{0, 1, 2, \ldots, k\}$. Let $\mathcal{F}_{\Gamma,k}$ be the set of all such k-characteristic functions on Γ. A function $f \in \mathcal{F}_{\Gamma,k}$ is a k-characteristic function for a set $S \subseteq \Gamma$ if $f(a) \in \{1, 2, \ldots, k\}$, for all $a \in S$, and $f(a) = 0$, for all $a \notin S$.

An *ordered-dataset* $\mathfrak{U}$ over the alphabet Γ consists of a finite set U, in which each element $u \in U$ carries a label $\ell ab_{\mathfrak{U}}(u) \in \Gamma$ and a data value $va\ell_{\mathfrak{U}}(u) \in \mathbb{N}$. An element $u \in U$ is called an a-element, if $\ell ab_{\mathfrak{U}}(u) = a$. In other words, an ordered-dataset is similar to an ordered-data tree, but without the relations $E_{\downarrow}$ and $E_{\rightarrow}$. It can be viewed as a structure $\mathfrak{U} = \langle U, \{a(\cdot)\}_{a\in\Gamma}, \sim, \prec, \prec_{suc}\rangle$, where

—for each $a \in \Gamma$ and $u \in U$, the relation $a(u)$ holds if $\ell ab_{\mathfrak{U}}(u) = a$,
—$u \sim v$ holds, if $va\ell_{\mathfrak{U}}(u) = va\ell_{\mathfrak{U}}(v)$,
—$u \prec v$ holds, if $va\ell_{\mathfrak{U}}(u) < va\ell_{\mathfrak{U}}(v)$,
—$u \prec_{suc} v$ holds, if $va\ell_{\mathfrak{U}}(v)$ is the minimal data value found in $\mathfrak{U}$ greater than $va\ell_{\mathfrak{U}}(u)$.

Let $\mathfrak{U}$ be an ordered-dataset and $d_1 < \cdots < d_m$ be the data values found in $\mathfrak{U}$. The *k-extended representation* of $\mathfrak{U}$ is the string $\mathcal{V}^k_\Gamma(\mathfrak{U}) = (S_1, f_1)\cdots(S_m, f_m) \in 2^\Gamma \times \mathcal{F}_{\Gamma,k}$ such that $S_1 \cdots S_m = \mathcal{V}_\Gamma(\mathfrak{U})$ and for each $i \in \{1, 2, \ldots, m\}$ and for each $a \in \Gamma$,

(1) f_i is a k-characteristic function for the set S_i,
(2) if $1 \leq f_i(a) \leq k - 1$, then there are $f_i(a)$ number of a-elements in U with data value d_i,
(3) if $f_i(a) = k$, then there are at least k number of a-elements in U with data value d_i.

We assume that in every formula in MSO$(\sim, \prec, \prec_{suc})$, all the monadic second-order quantifiers precede the first-order part. That is, sentences in MSO$(\sim, \prec, \prec_{suc})$ are of the form: $\varphi := Q_1X_1 \cdots Q_sX_s\ \psi$, where the X_i's are monadic second-order variables, the Q_i's are $\exists$ or $\forall$ and $\psi \in$ FO$(\sim, \prec, \prec_{suc})$ extended with the unary predicates $X_1, \ldots, X_s$. We call the integer s, the MSO quantifier rank of φ, denoted by $\mathsf{MSO\text{-}qr}(\varphi) = s$, while we write $\mathsf{FO\text{-}qr}(\varphi)$ to denote the quantifier rank of ψ, that is, the quantifier rank of the first-order part of φ.

LEMMA 4.2. *Let $\mathfrak{U}_1$ and $\mathfrak{U}_2$ be ordered-datasets over Γ such that $\mathcal{V}^{k2^s}_\Gamma(\mathfrak{U}_1) = \mathcal{V}^{k2^s}_\Gamma(\mathfrak{U}_2)$. For any MSO$(\sim, \prec, \prec_{suc})$ sentence φ such that $\mathsf{MSO\text{-}qr}(\varphi) \leq s$ and $\mathsf{FO\text{-}qr}(\varphi) \leq k$, $\mathfrak{U}_1 \models \varphi$ if and only if $\mathfrak{U}_2 \models \varphi$.*

PROOF. The proof is by Ehrenfeucht-Fraïssé game for MSO of $(s + k)$ rounds, with s rounds of set-moves and k rounds of point-moves. We can assume that the set-moves precede the point-moves (see, e.g., [Libkin 2004] for the definition of Ehrenfeucht-Fraïssé game).

Before we go to the proof, we need a few notations. Let $\mathfrak{U}_1$ and $\mathfrak{U}_2$ be ordered-datasets over Γ. For $(a, d) \in \Gamma \times \mathbb{N}$, we write $P_{\mathfrak{U}_1}(a, d) = \{u \mid \ell ab_{\mathfrak{U}_1}(u) = a \text{ and } va\ell_{\mathfrak{U}_1}(u) = d\}$ – the set of elements in U_1 with label a and data value d. We can define similarly $P_{\mathfrak{U}_2}(a, d)$ for $\mathfrak{U}_2$.

Let $\mathcal{O} \subseteq \{\sim, \prec, \prec_{suc}\}$. Let $u_1, \ldots, u_k \in U_1$ and $v_1, \ldots, v_k \in U_2$, for some ordered-datasets U_1 and U_2. The mapping $(u_1, \ldots, u_k) \mapsto (v_1, \ldots, v_k)$ is a partial $\mathcal{O}$-isomorphism

(with equality) from U_1 to U_2, if it is a partial isomorphism with regards to the vocabulary $\mathcal{O}$, and if $u_l = u_{l'}$, then $v_l = v_{l'}$.

We are going to describe Duplicator's strategy for winning the Ehrenfeucht-Fraïssé game for MSO of s rounds of set-moves, followed by k rounds of point moves. We start with the set-moves.

Duplicator's Strategy for Set-Moves. Suppose that the game is already played for l rounds, where $X_1, \ldots, X_l$ and $Y_1, \ldots, Y_l$ are the sets of positions chosen in U_1 and U_2, respectively. For each $I \subseteq \{1, \ldots, l\}$, define the following set:

$$P_{\mathfrak{U}_1}(a, d; I) \;=\; P_{\mathfrak{U}_1}(a, d) \cap \bigcap_{i \in I} X_i \cap \bigcap_{j \notin I} \overline{X_j},$$

$$P_{\mathfrak{U}_2}(a, d; I) \;=\; P_{\mathfrak{U}_2}(a, d) \cap \bigcap_{i \in I} Y_i \cap \bigcap_{j \notin I} \overline{Y_j}.$$

Duplicator's strategy is to preserve the following identity: for every $(a, d) \in \Gamma \times \mathbb{N}$ and every $I \subseteq \{1, \ldots, l\}$.

—If the cardinality $|P_{\mathfrak{U}_1}(a, d; I)| < k2^{m-l}$, then $|P_{\mathfrak{U}_1}(a, d; I)| = |P_{\mathfrak{U}_2}(a, d; I)|$.
—If the cardinality $|P_{\mathfrak{U}_1}(a, d; I)| \geq k2^{m-l}$, then also $|P_{\mathfrak{U}_2}(a, d; I)| \geq k2^{m-l}$.

Now suppose that on the $(l+1)^{\text{th}}$ set-move, Spoiler chooses a set X of positions on U_1. Duplicator chooses a set Y of positions on U_2 as follows. For each $I \subseteq \{1, \ldots, l\}$, there are four cases.

(1) $|P_{\mathfrak{U}_1}(a, d; I) \cap X|$ and $|P_{\mathfrak{U}_1}(a, d; I) \cap \overline{X}| < k2^{m-l-1}$. Then, $|P_{\mathfrak{U}_1}(a, d; I)| < k2^{m-l}$, which by induction hypothesis, implies $|P_{\mathfrak{U}_2}(a, d; I)| = |P_{\mathfrak{U}_1}(a, d; I)|$. Duplicator picks $|P_{\mathfrak{U}_1}(a, d; I) \cap X|$ number of points from $P_{\mathfrak{U}_2}(a, d; I)$, and declares them to "belong to Y." The rest of the points from $P_{\mathfrak{U}_2}(a, d; I)$ are declared to "not belong to Y."
Obviously, $|P_{\mathfrak{U}_1}(a, d; I) \cap X| = |P_{\mathfrak{U}_2}(a, d; I) \cap Y| < k2^{m-l-1}$ and $|P_{\mathfrak{U}_1}(a, d; I) \cap \overline{X}| = |P_{\mathfrak{U}_2}(a, d; I) \cap \overline{Y}| < k2^{m-l-1}$.

(2) $|P_{\mathfrak{U}_1}(a, d; I) \cap X| < k2^{m-l-1}$ and $|P_{\mathfrak{U}_1}(a, d; I) \cap \overline{X}| \geq k2^{m-l-1}$. In this case, either $P_{\mathfrak{U}_1}(a, d; I) < k2^m$ or $\geq k2^m$. In either case there are $|P_{\mathfrak{U}_1}(a, d; I) \cap X|$ number of points from $P_{\mathfrak{U}_2}(a, d; I)$ which Duplicator declares to "belong to Y." The rest of the points from $P_{\mathfrak{U}_2}(a, d; I)$ are declared to "not belong to Y."
Obviously, $|P_{\mathfrak{U}_1}(a, d; I) \cap X| = |P_{\mathfrak{U}_2}(a, d; I) \cap Y|$ and $|P_{\mathfrak{U}_2}(a, d; I) \cap \overline{Y}| \geq k2^{m-l-1}$.

(3) $|P_{\mathfrak{U}_1}(a, d; I) \cap X| \geq k2^{m-l-1}$ and $|P_{\mathfrak{U}_1}(a, d; I) \cap \overline{X}| < k2^{m-l-1}$. Similar to Case 2.

(4) $|P_{\mathfrak{U}_1}(a, d; I) \cap X| \geq k2^{m-l-1}$ and $|P_{\mathfrak{U}_1}(a, d; I) \cap \overline{X}| \geq k2^{m-l-1}$. Then, $|P_{\mathfrak{U}_1}(a, d; I)| \geq k2^{m-l}$, and so $|P_{\mathfrak{U}_2}(a, d; I)| \geq k2^{m-l}$. Duplicator declares half of the points in $P_{\mathfrak{U}_2}(a, d; I)$ to "belong to Y" and the other half to "not belong to Y."
Obviously, $|P_{\mathfrak{U}_2}(a, d; I) \cap Y|$ and $|P_{\mathfrak{U}_2}(a, d; I) \cap \overline{Y}| \geq k2^{m-l-1}$.

Now after m rounds of set-moves, we have the following identity: for every $(a, d) \in \Sigma \times \mathbb{N}$ and every $I \subseteq \{1, \ldots, m\}$.

—If the cardinality $|P_{\mathfrak{U}_1}(a, d; I)| < k$, then $|P_{\mathfrak{U}_1}(a, d; I)| = |P_{\mathfrak{U}_2}(a, d; I)|$.
—If the cardinality $|P_{\mathfrak{U}_1}(a, d; I)| \geq k$, then also $|P_{\mathfrak{U}_2}(a, d; I)| \geq k$.

This ends our description of Duplicator's strategy for set-moves. Now we describe Duplicator's strategy for point-moves.

Duplicator's Strategy for Point-Moves. Suppose that the game is now on lth step. Let $(u_1, \ldots, u_l) \mapsto (v_1, \ldots, v_l)$ be a partial $\{\sim, \prec, \prec_{suc}\}$-isomorphism, where $0 \leq l \leq k-1$. Suppose Spoiler chooses an element u_{l+1} from U_1 such that $va\ell_{\mathfrak{U}_1}(u_{l+1})$ is the j^{th} largest data value in $\mathfrak{U}_1$.

—If $u_{l+1} = u_{l'}$, for some $l' \in \{1, \ldots, l\}$, Duplicator chooses $v_{l+1} = v_{l'}$ from U_2.
—If $u_{l+1} \notin \{u_1, \ldots, u_l\}$, Duplicator chooses v_{l+1} from U_2 such that $v_{l+1} \notin \{v_1, \ldots, v_l\}$ and $\ell ab_{\mathfrak{U}_1}(u_{l+1}) = \ell ab_{\mathfrak{U}_2}(v_{l+1})$ and $va\ell_{\mathfrak{U}_2}(v_{l+1})$ is the j^{th} largest data value in $\mathfrak{U}_2$. Such an element exists, as $\mathcal{V}^{k2^m}(\mathfrak{U}_1) = \mathcal{V}^{k2^m}(\mathfrak{U}_2)$.

In either case, $(u_1, \ldots, u_{l+1}) \mapsto (v_1, \ldots, v_{l+1})$ is a partial $\{\sim, \prec, \prec_{suc}\}$-isomorphism. This completes the description of Duplicator's strategy and hence, our proof. □

Now, we define the k-extended representation of an ordered-data tree t over the alphabet Γ, denoted by $\mathcal{V}^k_\Gamma(t)$ is the k-extended representation of the ordered-dataset $\mathfrak{U}$ obtained by ignoring the relations $E_\downarrow$ and $E_\rightarrow$ in t. The following corollary is an immediate consequence of Lemma 4.2.

COROLLARY 4.3. *Let t_1 and t_2 be ordered-data trees over Γ such that $\mathcal{V}^{k2^s}_\Gamma(t_1) = \mathcal{V}^{k2^s}_\Gamma(t_2)$. For any MSO$(\sim, \prec, \prec_{suc})$ sentence φ such that MSO-qr$(\varphi) \le s$ and FO-qr$(\varphi) \le k$, $t_1 \models \varphi$ if and only if $t_2 \models \varphi$.*

PROOF. Since the predicates $E_\downarrow$ and $E_\rightarrow$ are not used in the formula $\varphi \in \text{MSO}(\sim, \prec, \prec_{suc})$, we can ignore them in t_1 and t_2 and view both t_1 and t_2 as ordered-datasets. Our corollary follows immediately from Lemma 4.2. □

5. AUTOMATA FOR ORDERED-DATA TREE

In this section, we are going to introduce an automata model for ordered-data trees and study its expressive power.

Definition 5.1. An ordered-data tree automaton, in short ODTA, over the alphabet Σ is a triplet $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$, where $\mathcal{T}$ is a letter-to-letter nondeterministic transducer from $\Sigma \times \{\top, \bot, *\}^3$ to the output alphabet Γ; $\mathcal{M}$ is an automaton on strings over the alphabet 2^Γ; and $\Gamma_0 \subseteq \Gamma$.

An ordered-data tree t is accepted by $\mathcal{S}$, denoted by $t \in \mathcal{L}_{data}(\mathcal{S})$, if there exists an ordered-data tree t' over Γ such that

—on input Profile(t), the transducer $\mathcal{T}$ outputs t';
—the automaton $\mathcal{M}$ accepts the string $\mathcal{V}_\Gamma(t')$; and
—for every $a \in \Gamma_0$, all the a-nodes in t' have different data values.

We describe a few examples of ODTA that accept the languages described in Examples 3.3, 3.4, 3.5, and 3.6.

Example 5.2. An ODTA $\mathcal{S}^a = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ that accepts the language $\mathcal{L}^a_{data}$ in Example 3.3 can be defined as follows. The output alphabet of the transducer $\mathcal{T}$ is $\Gamma = \{\alpha, \beta, \gamma\}$. On an input tree t, the transducer $\mathcal{T}$ marks the nodes in t as follows. There is only one node marked with α, one node marked with β, and the α-node is an ancestor of β. The automaton $\mathcal{M}$ accepts all the strings in which the position labeled with $S \ni \beta$ is less than or equal to the position labeled with $S' \ni \alpha$. (These two positions can be equal, which means $S = S'$.) Finally, $\Gamma_0 = \emptyset$.

Example 5.3. An ODTA $\mathcal{S}^{S,m} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ that accepts the language $\mathcal{L}^{S,m}_{data}$ in Example 3.4 can be defined as follows. The transducer $\mathcal{T}$ is an identity transducer. The automaton $\mathcal{M}$ accepts all the strings in which the symbol S appears exactly m times, and $\Gamma_0 = \emptyset$.

Example 5.4. An ODTA $\mathcal{S}^{S,\,(\text{mod } m)} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ that accepts the language $\mathcal{L}^{S,\,(\text{mod } m)}_{data}$ in Example 3.5 can be defined as follows. The transducer $\mathcal{T}$ is an identity

transducer. The automaton $\mathcal{M}$ accepts a string in which the number of appearances of the symbol S is a multiple of m, and $\Gamma_0 = \emptyset$.

Example 5.5. An ODTA $\mathcal{S}^{a*} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ that accepts the language $\mathcal{L}^{a*}_{data}$ in Example 3.6 can be defined as follows. The output alphabet of the transducer $\mathcal{T}$ is $\Gamma = \{\alpha, \beta\}$. The transducer $\mathcal{T}$ marks the nodes as follows. A node is marked with α if and only if it is an a-node and it has different data value from the one of its parent. All the other nodes are marked with β. The automaton $\mathcal{M}$ accepts a string v if and only if the last symbol in v contains the symbol α, while $\Gamma_0 = \{\alpha\}$.

The following proposition states that ODTA languages are closed under union and intersection but not under negation. We would like to remark that being not closed under negation is rather common for decidable models for data trees. Often models that are closed under negation have undecidable non-emptiness/satisfiability problem.

PROPOSITION 5.6. *The class of languages accepted by ODTA is closed under union and intersection but not under negation.*

PROOF. For closure under union and intersection, let $\mathcal{S}_1 = \langle \mathcal{T}_1, \mathcal{M}_1, \Gamma^1_0 \rangle$ and $\mathcal{S}_2 = \langle \mathcal{T}_2, \mathcal{M}_2, \Gamma^2_0 \rangle$ be ODTA. The union $\mathcal{L}_{data}(\mathcal{S}_1) \cup \mathcal{L}_{data}(\mathcal{S}_2)$ is accepted by an ODTA which non-deterministically chooses to simulate either $\mathcal{S}_1$ or $\mathcal{S}_2$ on the input ordered-data tree. The ODTA for the intersection $\mathcal{L}_{data}(\mathcal{S}_1) \cap \mathcal{L}_{data}(\mathcal{S}_2)$ can be obtained by the standard cross product between $\mathcal{S}_1$ and $\mathcal{S}_2$.

We now prove hat ODTA languages are not closed under negation. Consider the negation of the language in Example 3.3, whose equivalent ODTA $\mathcal{S}^a$ is presented in Example 5.2. Every tree $t \notin \mathcal{L}(\mathcal{S}^a)$ has the following property. If u, v are two a-nodes in t and u is an ancestor of v, then $u \prec v$.

Now suppose to the contrary that there exists an ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ that accepts the negation of $\mathcal{L}(\mathcal{S}^a)$. Let Γ be the output alphabet of $\mathcal{T}$. Let $t \in \mathcal{L}(\mathcal{S})$ be a data tree with $|\Gamma| + 1$ nodes, where each node is labeled with a and has at most one child. This implies that the data values in t are all different and appear in increasing order from the root node to the leaf node.

Let $t' \in \mathcal{T}(t)$. Since t has $|\Gamma| + 1$ nodes, and hence so does t', there are two nodes in u and v in t' with the same label. Let t'' be a data tree obtained from t by swapping the data values between u and v, so $t'' \in \mathcal{L}(\mathcal{S}^a)$. Since $\mathsf{Profile}(t) = \mathsf{Profile}(t'')$, on input $\mathsf{Profile}(t'')$, the transducer $\mathcal{T}$ can also output t', which means that $t'' \in \mathcal{L}(\mathcal{S})$. This contradicts the fact that $\mathcal{L}(\mathcal{S})$ is the complement of $\mathcal{L}(\mathcal{S}^a)$. This completes the proof of Proposition 5.6. □

We should remark that in Section 7, we will discuss that extending ODTA with the complement of languages of the form in Example 5.2 will immediately yield undecidability.

Theorems 5.7, 5.8, and 5.9 are the main results in this article. Theorem 5.7 provides the ODTA characterisation of the logic $\exists\mathrm{MSO}^2(E_\downarrow, E_\rightarrow, \sim)$, and its proof can be found in Section 5.1.

THEOREM 5.7. *A language $\mathcal{L}_{data}$ is expressible with an $\exists MSO^2(E_\downarrow, E_\rightarrow, \sim)$ formula if and only if it is accepted by an ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$, where $\mathcal{L}(\mathcal{M})$ is a commutative language. Moreover, the translation from $\exists MSO^2(E_\downarrow, E_\rightarrow, \sim)$ formulas to ODTA takes triple exponential time, while from ODTA to $\exists MSO^2(E_\downarrow, E_\rightarrow, \sim)$ formulas, takes exponential time.*

Theorem 5.8 provides the logical characterisation of ODTA. The proof can be found in Section 5.2.

THEOREM 5.8. *A language $\mathcal{L}_{data}$ is accepted by an ODTA if and only if it is expressible with a formula of the form: $\exists X_1 \cdots \exists X_m\, \varphi \wedge \psi$, where φ is a formula from $FO^2(E_{\downarrow}, E_{\rightarrow}, \sim)$, and ψ from $FO(\sim, \prec, \prec_{suc})$, both extended with the unary predicates $X_1, \ldots, X_m$ and $a(\cdot)$. Moreover, the translation from ODTA to formula is of polynomial time, and from formula to ODTA is effective, but non-elementary.*

Finally, we show that the non-emptiness problem for ODTA is decidable in Theorem 5.9. The proof can be found in Section 5.3.

THEOREM 5.9. *The non-emptiness problem for ODTA is decidable in 3-*NEXPTIME.

The best lower bound known up to date is NP-hard. See Fan and Libkin [2002], David et al. [2012].

5.1. Proof of Theorem 5.7

In the proof, we assume that the ordered-data trees are over the finite alphabet Σ. We will need the following proposition which states that every $\exists\mathrm{MSO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$ formula can be syntactically rewritten to a normal form for $\exists\mathrm{MSO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$.

PROPOSITION 5.10 ([BOJANCZYK ET AL. 2009, PROPOSITION 3.8]). *Every formula $\psi \in \exists MSO^2(E_{\downarrow}, E_{\rightarrow}, \sim)$ can be rewritten into a normal form of exponential size of the form: $\exists Y_1 \cdots \exists Y_n\, \varphi$, where φ is a conjunction of formulae of the form:*

(N1) $\forall x \forall y\, (\alpha(x) \wedge \delta(x, y) \wedge \xi(x, y) \rightarrow \beta(y))$,
(N2) $\forall x\, (\mathrm{root}(x) \rightarrow \alpha(y))$,
(N3) $\forall x\, (\text{first-sibling}(x) \rightarrow \alpha(y))$,
(N4) $\forall x\, (\text{last-sibling}(x) \rightarrow \alpha(y))$,
(N5) $\forall x\, (\mathrm{leaf}(x) \rightarrow \alpha(y))$,
(N6) $\forall x \forall y\, (\alpha(x) \wedge \alpha(y) \wedge x \sim y \rightarrow x = y)$,
(N7) $\forall x \exists y\, (\alpha(x) \rightarrow \beta(y) \wedge x \sim y)$,

where $\alpha(x), \beta(x)$ is a conjunction of some unary predicates and its negations, $\delta(x, y)$ is either $E_{\downarrow}(x, y)$ or $E_{\rightarrow}(x, y)$, and $\xi(x, y)$ is either $x \sim y$ or $x \nsim y$.

We should remark that if φ is a conjunction of formulae of the forms (N1)–(N5), then there exists a tree automaton $\mathcal{A}$ over the alphabet $\Sigma \times \{\top, \bot, *\}^3$ such that for every ordered-data tree t,

$$t \models \Psi \quad \text{if and only if} \quad \mathsf{Profile}(t) \text{ is accepted by } \mathcal{A}.$$

Such construction is straightforward from the classical automata theory (see, e.g., [Thomas 1997]). We next divide the proof of Theorem 5.7 into Lemmas 5.11 and 5.12.

LEMMA 5.11. *For every formula $\Psi \in \exists MSO^2(E_{\downarrow}, E_{\rightarrow}, \sim)$, there exists an ODTA $\mathcal{S}_\Psi = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ such that $\mathcal{L}_{data}(\Psi) = \mathcal{L}_{data}(\mathcal{S}_\Psi)$ and $\mathcal{L}(\mathcal{M})$ is commutative. Moreover, the construction of $\mathcal{S}_\Psi$ is effective and takes triple exponential time in the size of the formula Ψ.*

PROOF. Applying Proposition 5.10, we can rewrite the formula Ψ in its normal form $\exists Y_1 \cdots \exists Y_n \Psi'$. Furthermore, we can rewrite the formula Ψ into the form $\exists X_1 \cdots \exists X_m\, \varphi$, where $m = 2^n$, and φ is a conjunction of formulas of the following form.

(N0′) $X_1, \ldots, X_m$ are pairwise disjoint, and $\bigwedge_{a \in \Sigma} \forall x (a(x) \rightarrow \alpha'(x))$.
(N1′) $\forall x \forall y\, (\alpha'(x) \wedge \delta(x, y) \wedge \xi(x, y) \rightarrow \beta'(y))$,
(N2′) $\forall x\, (\mathrm{root}(x) \rightarrow \alpha'(y))$,
(N3′) $\forall x\, (\text{first-sibling}(x) \rightarrow \alpha'(y))$,
(N4′) $\forall x\, (\text{last-sibling}(x) \rightarrow \alpha'(y))$,

(N5′) $\forall x\ (\mathrm{leaf}(x) \rightarrow \alpha'(y))$,
(N6′) $\forall x \forall y\ (\alpha'(x) \wedge \alpha'(y) \wedge x \sim y \rightarrow x = y)$,
(N7′) $\forall x \exists y\ (\alpha'(x) \rightarrow \beta'(y) \wedge x \sim y)$,

where $\alpha'(x), \beta'(x)$ are disjunctions of some of the X_i's, and $\delta(x, y)$ and $\xi(x, y)$ are the same as before. Intuitively, the unary predicates $X_1, \ldots, X_m$ correspond to subsets of $\{Y_1, \ldots, Y_n\}$.

The ODTA $\mathcal{S}_\Psi = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ is defined as follows.

—The transducer $\mathcal{T}$ checks whether the formulas (N0′)–(N5′) are satisfied, with the output alphabet $\Gamma = \{X_1, \ldots, X_m\}$, where a node is labeled with X_i if and only if it belongs to X_i.
The construction of such transducer is straightforward, thus, omitted (see, e.g., [Thomas 1997]).
—Γ_0 consists of the X_i's, where there exists $A \subseteq \{X_1, \ldots, X_m\}$ and $X_i \in A$ and a formula of the form (N6′)

$$\forall x \forall y \left(\bigvee_{X_j \in A} X_j(x) \wedge \bigvee_{X_j \in A} X_j(y) \wedge x \sim y \rightarrow x = y \right),$$

in φ.
—The automaton $\mathcal{M}$ accepts the language $(2^{\{X_1,\ldots,X_m\}} - (\mathcal{P}_1 \cup \mathcal{P}_2))^*$, where

$$\mathcal{P}_1 := \left\{ S \;\middle|\; \begin{array}{l} \text{there exists a formula} \\ \quad \forall x \exists y\ (\bigvee_{X \in A} X(x) \rightarrow \bigvee_{X \in B} X(y) \wedge x \sim y) \\ \text{in } \varphi \text{ such that } S \cap A \neq \emptyset \text{ but } S \cap B = \emptyset \end{array} \right\},$$

$$\mathcal{P}_2 := \left\{ S \;\middle|\; \begin{array}{l} \text{there exists a formula} \\ \quad \forall x \forall y\ (\bigvee_{X \in A} X(x) \wedge \bigvee_{X \in A} X(y) \wedge x \sim y \rightarrow x = y) \\ \text{in } \varphi \text{ such that } |S \cap A| \geq 2 \end{array} \right\}.$$

That $\mathcal{L}(\mathcal{M})$ is commutative is trivial. That $\mathcal{S}$ accepts precisely the language $\mathcal{L}_{data}(\Psi)$ can be deduced from the following.

—That $\mathcal{T}$ ensures that formulas N0′–N5′ are satisfied.
—That Γ_0 contains precisely the symbols X_i's, where all X_i-nodes are supposed to contain different data values.
—That for every ordered-data tree t,

$$t \models \forall x \exists y \left(\bigvee_{X \in A} X(x) \rightarrow \bigvee_{X \in B} X(y) \wedge x \sim y \right),$$

if and only if $[S]_t = \emptyset$ for all S such that $S \cap A \neq \emptyset$ but $S \cap B = \emptyset$.
—That for every ordered-data tree t,

$$t \models \forall x \forall y \left(\bigvee_{X \in A} X(x) \wedge \bigvee_{X \in A} X(y) \wedge x \sim y \rightarrow x = y \right),$$

if and only if the following hold.
—$[S]_t = \emptyset$ for all S such that $|S \cap A| \geq 2$.
—For all $X \in A$, $t \models \forall x \forall y\ (X(x) \wedge X(y) \wedge x \sim y \rightarrow x = y)$, which is captured by the condition imposed by Γ_0.

The analysis of the complexity is as follows. The first step, applying Proposition 5.10, induces an exponential blow-up in the size of the input. The second step to construct the formula $\exists X_1 \cdots \exists X_m\ \varphi$ takes exponential time in n, and n is exponential in the size of the input. The construction of $\mathcal{T}$ takes polynomial time in the size of φ, since

(N0′)–(N5′) are already in the "automata transition" format. The construction of Γ_0 takes polynomial time in m, while the construction of $\mathcal{M}$ induces another exponential blow-up in m. Altogether, the complexity of our constructing $\mathcal{S}_\Psi$ is triple exponential time in the size of Ψ. This concludes the proof of Lemma 5.11. □

For the complexity analysis in Lemma 5.12, we assume that a commutative automaton $\mathcal{M}$ is given as a set of vectors (in binary format) indicating its Parikh images. That is, $\mathcal{M}$ is given as a set $I = \{(\bar{u}_1, \bar{v}_{1,1}, \ldots, \bar{v}_{1,\ell}), \ldots, (\bar{u}_n, \bar{v}_{n,1}, \ldots, \bar{v}_{n,\ell})\}$, where

$$\bigcup_{(\bar{u},\bar{v}_1,\ldots,\bar{v}_\ell)\in I} \mathcal{L}(\bar{u}, \bar{v}_1, \ldots, \bar{v}_\ell) = \mathcal{L}(\mathcal{M}),$$

and each number in the vectors in I is written in the standard binary form.

LEMMA 5.12. *For every ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$, where $\mathcal{L}(\mathcal{M})$ is a commutative language, there exists a formula $\varphi \in \exists MSO^2(E_\downarrow, E_\rightarrow, \sim)$ such that $\mathcal{L}_{data}(\varphi) = \mathcal{L}_{data}(\mathcal{S})$. Moreover, the construction of φ takes exponential time in the size of $\mathcal{S}$.*

PROOF. Let $Q_\mathcal{T} = \{q_0, \ldots, q_m\}$ and $\Gamma = \{\alpha_1, \ldots, \alpha_k\}$ be the set of states and the output alphabet of the transducer $\mathcal{T}$, respectively. Let $\ell = 2^{|\Gamma|} - 1$.

By Theorem 2.1, $\mathcal{L}(\mathcal{M})$ is a finite union of periodic languages. Let I be the finite set of $(\ell+1)$-tuple of $\mathbb{N}^\ell$-vectors such that

$$\bigcup_{(\bar{u},\bar{v}_1,\ldots,\bar{v}_\ell)\in I} \mathcal{L}(\bar{u}, \bar{v}_1, \ldots, \bar{v}_\ell) = \mathcal{L}(\mathcal{M}).$$

Let $I = \{(\bar{u}_1, \bar{v}_{1,1}, \ldots, \bar{v}_{1,\ell}), \ldots, (\bar{u}_n, \bar{v}_{n,1}, \ldots, \bar{v}_{n,\ell})\}$ and $S_1, \ldots, S_\ell$ be the enumeration of nonempty subsets of Γ. First, for $(\bar{u}, \bar{v}_1, \ldots, \bar{v}_\ell) \in I$, we construct an $\exists \mathrm{MSO}^2(E_\downarrow, E_\rightarrow, \sim)$ formula $\Psi_{(\bar{u},\bar{v}_1,\ldots,\bar{v}_\ell)}$, where

$$t \in \Psi_{(\bar{u},\bar{v}_1,\ldots,\bar{v}_\ell)} \text{ if and only if } \left[\begin{array}{l}\text{there exists } h_1, \ldots, h_\ell \geq 0 \text{ such that} \\ (|[S_1]_t|, \ldots, |[S_\ell]_t|) = \bar{u} + h_1\bar{v}_1 + \cdots + h_\ell\bar{v}_\ell\end{array}\right].$$

We denote by v_i the nonzero entry of $\bar{v}_i$. This formula $\Psi_{(\bar{u},\bar{v}_1,\ldots,\bar{v}_\ell)}$ is as follows.

$$\begin{aligned}
&\exists W_{1,1} \cdots W_{1,u_1} \;\cdots\cdots\; \exists W_{\ell,1} \cdots W_{\ell,u_\ell} \\
&\quad \exists X_{1,0} \cdots X_{1,v_1-1}\, \exists Y_{1,0} \cdots Y_{1,v_1-1}\, Z_1 \\
&\qquad \ddots \\
&\quad \exists X_{\ell,0} \cdots X_{\ell,v_\ell-1}\, \exists Y_{\ell,0} \cdots Y_{\ell,v_\ell-1}\, Z_\ell \\
&\qquad \bigwedge_i W_{i,1}, \ldots, W_{i,u_i} \cap Z_i = \emptyset \\
&\qquad \wedge \bigwedge_i \varphi_{|[S_i]|=u_i}(W_{i,1}, \ldots, W_{i,u_i}) \\
&\qquad \wedge \bigwedge_i \varphi_{|[S_i]|\equiv v_i \pmod m}(X_{i,0}, \ldots, X_{i,v_i-1}, Y_{i,0}, \ldots, Y_{i,v_i-1}, Z_i),
\end{aligned}$$

where $\varphi_{S_i,u_i}(W_{i,1}, \ldots, W_{i,u_i})$ and $\varphi_{S_i,\,(\mathrm{mod}\ v_i)}(X_{i,0}, \ldots, X_{i,v_i-1}, Y_{i,0}, \ldots, Y_{i,v_i-1}, Z_i)$ are the formulas for the languages $\mathcal{L}_{data}^{S_i,u_i}$ and $\mathcal{L}_{data}^{S_i,\,(\mathrm{mod}\ u_i)}$ in Examples 3.4 and 3.5, respectively.

The desired formula φ is

$$\begin{aligned}
\exists X_{q_0} \cdots \exists X_{q_m}\, \exists X_{\alpha_1} \cdots \exists X_{\alpha_k}\, \exists \overline{X}_{(\bar{u}_1,\bar{v}_{1,1},\ldots,\bar{v}_{1,\ell})} \;\cdots\; \exists \overline{X}_{(\bar{u}_n,\bar{v}_{n,1},\ldots,\bar{v}_{n,\ell})} \\
\varphi_{\Gamma_0} \wedge \varphi_\mathcal{T} \wedge \bigvee_{(\bar{u},\bar{v}_1,\ldots,\bar{v}_\ell)\in I} \varphi_{(\bar{u},\bar{v}_1,\ldots,\bar{v}_\ell)},
\end{aligned}$$

where the following hold.

—The formula φ_{Γ_0} expresses the fact that the data values found under nodes labeled with a symbol from Γ_0 are all different.
—The unary predicates $X_{q_0}, \ldots, X_{q_m}, X_{\alpha_1}, \ldots, X_{\alpha_k}$ are supposed to represent the states and the output alphabets of $\mathcal{T}$, respectively.
—The formula $\varphi_{\mathcal{T}}$ expresses the behaviour of the transducer $\mathcal{T}$, that is, a tree satisfies $\varphi_{\mathcal{T}}$ in which for every node $u \in \mathsf{Dom}(t)$, $X_{q_i}(u)$ and $X_{\alpha_j}(u)$ holds, if there exists an accepting run of $\mathcal{T}$ on t in which the node u is labeled with q_i and output α_j.
—The predicates $\overline{X}_{(\bar{u_i}, \bar{v}_{i,1}, \ldots, \bar{v}_{i,\ell})}$'s and the formulas $\varphi_{(\bar{u_i}, \bar{v}_{i,1}, \ldots, \bar{v}_{i,\ell})}$'s are as in the formula $\Psi_{(\bar{u}, \bar{v}_1, \ldots, \bar{v}_\ell)}$ previously defined.

The analysis of the complexity is as follows. The size of the formula φ_{S_i, u_i} and $\varphi_{S_i, \pmod{v_i}}$ are exponential in the size of S_i, u_i, v_i. Hence, the construction of $\Psi_{(\bar{u}, \bar{v}_1, \ldots, \bar{v}_\ell)}$ takes exponential time in the size of $(\bar{u}, \bar{v}_1, \ldots, \bar{v}_\ell)$. The construction of φ_{Γ_0} and $\varphi_{\mathcal{T}}$ takes polynomial time in the size of Γ_0 and $\mathcal{T}$, respectively. Hence, the total time to construct the formula φ is exponential in the size of $\mathcal{S}$. This completes the proof of the lemma. □

5.2. Proof of Theorem 5.8

In this section, for every ordered-data tree t, we assume that the data values in t are precisely the natural numbers in the range $[1..m]$, for a positive integer $m \geq 1$. That is, if $d_1 < d_2 < \cdots < d_m$ are the data values in t, then $d_1 = 1, d_2 = 2, \ldots, d_m = m$.

We start with the following lemma.

LEMMA 5.13. *Let $\psi \in FO(\sim, \prec)$ be of quantifier rank k. Let $\Gamma = \{a_1, \ldots, a_\ell\}$ be the set of unary predicates used in ψ. There exists a finite state automaton C over the alphabet $\Gamma \cup (2^\Gamma \times \mathcal{F}_{\Gamma,k})$ such that the following holds.*

—*The automaton C accepts words of the form*

$$\overbrace{a_1 \cdots a_1}^{f_1(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_1(a_\ell)} (S_1, f_1) \cdots\cdots \overbrace{a_1 \cdots a_1}^{f_m(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_m(a_\ell)} (S_m, f_m),$$

where each $S_i = \{a \mid f_i(a) \geq 1\}$.
—*For every ordered-data tree $t \models \psi$, if $\mathcal{V}^{(k)} = (S_1, f_1), \ldots, (S_m, f_m)$, then there exists a word in $\mathcal{L}(C)$ of the form*

$$\overbrace{a_1 \cdots a_1}^{f_1(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_1(a_\ell)} (S_1, f_1) \cdots\cdots \overbrace{a_1 \cdots a_1}^{f_m(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_m(a_\ell)} (S_m, f_m).$$

—*For every word $w \in \mathcal{L}(C)$, if w is*

$$\overbrace{a_1 \cdots a_1}^{f_1(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_1(a_\ell)} (S_1, f_1) \cdots\cdots \overbrace{a_1 \cdots a_1}^{f_m(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_m(a_\ell)} (S_m, f_m),$$

then there exists a tree $t \models \psi$, where $\mathcal{V}^{(k)}(t) = (S_1, f_1) \cdots (s_m, f_m)$.

PROOF. Let $\psi \in \mathrm{FO}(\sim, \prec)$ be of quantifier rank k. Let $\Gamma = \{a_1, \ldots, a_\ell\}$ be the set of unary predicates used in φ. We define the following sentence $\overline{\psi} \in \mathrm{FO}(<)$ (i.e., over strings) inductively from ψ as follows.

—If ψ is $Qx\ \xi$, where $Q \in \{\forall, \exists\}$, then $\overline{\psi}$ is

$$Qx \bigvee_{a \in \Gamma} a(x) \to \overline{\xi}.$$

—If ψ is $x = y$, then $\overline{\psi}$ is also $x = y$.

—If ψ is $x \sim y$, then $\overline{\psi}$ states "there is no position in between x and y labeled with any symbol from $2^{\Gamma} \times \mathcal{F}_{\Gamma,k}$."
—If ψ is $x \prec y$, then $\overline{\psi}$ states "there is at least one position in between x and y labeled with a symbol from $2^{\Gamma} \times \mathcal{F}_{\Gamma,k}$."

We have the following claim.

CLAIM 1. *(1) For every ordered-data tree $t \models \psi$, if $\mathcal{V}^{(k)} = (S_1, f_1), \ldots, (S_m, f_m)$, then there exists a word $w \models \overline{\psi}$ of the form*

$$\overbrace{a_1 \cdots a_1}^{f_1(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_1(a_\ell)} (S_1, f_1) \cdots\cdots \overbrace{a_1 \cdots a_1}^{f_m(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_m(a_\ell)} (S_m, f_m).$$

(2) For every word $w \models \overline{\psi}$, if w is

$$\overbrace{a_1 \cdots a_1}^{f_1(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_1(a_\ell)} (S_1, f_1) \cdots\cdots \overbrace{a_1 \cdots a_1}^{f_m(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_m(a_\ell)} (S_m, f_m),$$

then there exists a tree $t \models \psi$, where $\mathcal{V}^{(k)}(t) = (S_1, f_1) \cdots (s_m, f_m)$.

PROOF. We first prove item (1). Let t be an ordered-data tree over the alphabet Γ and let $\mathcal{V}^k(t) = (S_1, f_1) \cdots (S_m, f_m)$ be its k-extended string representation of data values in t. Let t' be the following data string.

$$\overbrace{\binom{a_1}{1} \cdots \binom{a_1}{1}}^{f_1(a_1)} \cdots \overbrace{\binom{a_\ell}{1} \cdots \binom{a_\ell}{1}}^{f_1(a_\ell)} \cdots\cdots\cdots \overbrace{\binom{a_1}{m} \cdots \binom{a_1}{m}}^{f_m(a_1)} \cdots \overbrace{\binom{a_\ell}{m} \cdots \binom{a_\ell}{m}}^{f_m(a_\ell)}.$$

When t' is viewed as a data tree[4], $\mathcal{V}^{(k)}_{\Gamma}(t) = \mathcal{V}^{(k)}(t')$. Hence, by Corollary 4.3,

$$t \models \psi \qquad \text{if and only if} \qquad t' \models \psi.$$

By straightforward induction on $\overline{\psi}$, we can show that for every $t' \models \psi$ of the form

$$\overbrace{\binom{a_1}{1} \cdots \binom{a_1}{1}}^{f_1(a_1)} \cdots \overbrace{\binom{a_\ell}{1} \cdots \binom{a_\ell}{1}}^{f_1(a_\ell)} \cdots\cdots\cdots \overbrace{\binom{a_1}{m} \cdots \binom{a_1}{m}}^{f_m(a_1)} \cdots \overbrace{\binom{a_\ell}{m} \cdots \binom{a_\ell}{m}}^{f_m(a_\ell)},$$

there exists a word $w \models \overline{\psi}$ of the form

$$\overbrace{a_1 \cdots a_1}^{f_1(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_1(a_\ell)} (S_1, f_1) \cdots\cdots \overbrace{a_1 \cdots a_1}^{f_m(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_m(a_\ell)} (S_m, f_m).$$

Similarly, to prove (2), we can prove by straightforward induction on $\overline{\psi}$ that for every word $w \models \overline{\psi}$ of the form

$$\overbrace{a_1 \cdots a_1}^{f_1(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_1(a_\ell)} (S_1, f_1) \cdots\cdots \overbrace{a_1 \cdots a_1}^{f_m(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_m(a_\ell)} (S_m, f_m),$$

there exists a tree $t \models \psi$ of the form

$$\overbrace{\binom{a_1}{1} \cdots \binom{a_1}{1}}^{f_1(a_1)} \cdots \overbrace{\binom{a_\ell}{1} \cdots \binom{a_\ell}{1}}^{f_1(a_\ell)} \cdots\cdots\cdots \overbrace{\binom{a_1}{m} \cdots \binom{a_1}{m}}^{f_m(a_1)} \cdots \overbrace{\binom{a_\ell}{m} \cdots \binom{a_\ell}{m}}^{f_m(a_\ell)}$$

This completes the proof of our claim. □

[4]That is, a data string is a data tree in which each node has at most one child.

Let C be an automaton over the alphabet $\Gamma \cup (2^\Gamma \times \mathcal{F}_{\Gamma,k})$ that expresses the formula $\overline{\psi}$ and that it accepts only words of the form

$$\overbrace{a_1 \cdots a_1}^{f_1(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_1(a_\ell)} (S_1,\ f_1)\ \cdots\cdots\ \overbrace{a_1 \cdots a_1}^{f_m(a_1)} \cdots \overbrace{a_\ell \cdots a_\ell}^{f_m(a_\ell)} (S_m,\ f_m),$$

where each $S_i = \{a \mid f_i(a) \geq 1\}$. The construction of C from the formula $\overline{\psi}$ is rather standard, but non-elementary. See, for example, Thomas [1997]. That the automaton C is the desired automaton is immediate. This completes our proof of Lemma 5.13.

LEMMA 5.14. *Let $\psi \in FO(\sim, \prec)$ be of quantifier rank k. Let $\Gamma = \{a_1, \ldots, a_\ell\}$ be the set of unary predicates used in ψ. There exists a finite state automaton $\mathcal{M}$ over the alphabet $2^\Gamma \times \mathcal{F}_{\Gamma,k}$ such that $\mathcal{L}(\mathcal{M}) = \{\mathcal{V}^{(k)}_{\Gamma,k}(t) \mid t \models \psi\}$.*

PROOF. Let C be the automaton obtained by applying Lemma 5.13 on the formula ψ. Then let $\mathcal{M}$ be the automaton obtained from C, where every symbol from Γ is projected to empty string. The automaton $\mathcal{M}$ is the desired automaton, and this completes our proof of Lemma 5.14. □

Now we are ready to prove Theorem 5.8. We start with the "if" direction. Let Ψ be a formula of the form

$$\exists Y_1 \cdots \exists Y_n\ \varphi \wedge \psi,$$

φ is a formula from $\mathrm{FO}^2(E_\downarrow, E_\rightarrow, \sim)$ and ψ from $\mathrm{FO}(\sim, \prec)$, both extended with the unary predicates $Y_1, \ldots, Y_n$.

By Proposition 5.10, we can rewrite (with additional unary predicates) the formula φ into a conjunction of formulae of the form N1–N7 as stated in Proposition 5.10. Then we further rewrite it into the form

$$\exists X_1 \cdots \exists X_m\ \varphi' \wedge \psi',$$

where $m = 2^n$ and φ is a formula from $\mathrm{FO}^2(E_\downarrow, E_\rightarrow, \sim)$ and ψ from $\mathrm{FO}(\sim, \prec)$, both extended with the unary predicates $X_1, \ldots, X_m$, and that the formula φ' is conjunction of the following form.

(N0′) a formula ξ that states that $X_1, \ldots, X_m$ are pairwise disjoint and that

$$\bigwedge_{a \in \Sigma} \forall x\ (a(x) \rightarrow \alpha(x)),$$

(N1′) $\forall x \forall y\ (\alpha(x) \wedge \delta(x, y) \wedge \xi(x, y) \rightarrow \beta(y))$,
(N2′) $\forall x\ (\mathrm{root}(x) \rightarrow \alpha(y))$,
(N3′) $\forall x\ (\text{first-sibling}(x) \rightarrow \alpha(y))$,
(N4′) $\forall x\ (\text{last-sibling}(x) \rightarrow \alpha(y))$,
(N5′) $\forall x\ (\mathrm{leaf}(x) \rightarrow \alpha(y))$,
(N6′) $\forall x \forall y\ (\alpha(x) \wedge \alpha(y) \wedge x \sim y \rightarrow x = y)$,
(N7′) $\forall x \exists y\ (\alpha(x) \rightarrow \beta(y) \wedge x \sim y)$,

where $\alpha(x)$, $\beta(x)$ are disjunctions of some of the unary predicates $X_1, \ldots, X_m$.

We will describe the ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ for the formula Ψ, where the transducer $\mathcal{T}$ expresses the formula N0′–N5′ with the output alphabet $\Gamma = \{X_1, \ldots, X_m\}$, the automaton $\mathcal{M}$ expresses the formula N6′, N7′ and ψ', and Γ_0 is the set of symbols that appear in formula N6′. Formally, it is defined as follows.

—The output alphabet of $\mathcal{T}$ is $\Gamma = \{X_1, \ldots, X_m\}$.
—The transducer expresses the preceding formula N0′–N5′. In particular, the input and output symbols of each node must satisfy the formula N0′.

This step take polynomial time, since the formula N0′–N5′ is already in the transition format.

—The set $\Gamma_0 = \{X_i \mid X_i \text{ appears in N6}'\}$.
This step takes polynomial time.

—The automaton $\mathcal{M}$ expresses the formulas N6′, N7′, and ψ', obtained by applying Lemma 5.14.
This step is constructive but non-elementary due to the conversion from a formula to its finite state automaton.

It is straightforward to show that $\mathcal{L}_{data}(\mathcal{S}) = \{t \mid t \models \Psi\}$.
Now we prove the "only if" direction. Let $\mathcal{L} = \mathcal{L}_{data}(\mathcal{S})$, where $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$, and

—$Q = \{q_1, \ldots, q_n\}$ be the states of $\mathcal{T}$;
—$P = \{p_1, \ldots, p_s\}$ be the states of $\mathcal{M}$, and p_1 is the initial state of $\mathcal{M}$;
—$\Gamma = \{\alpha_1, \ldots, \alpha_\ell\}$ be the output alphabet of $\mathcal{T}$.

We denote by Σ the input alphabet of $\mathcal{T}$.
The desired formula for $\mathcal{L}$ is of the form:

$$\exists X_{q_1} \cdots \exists X_{q_n}\ \exists X_{\alpha_1} \cdots \exists X_{\alpha_\ell}\ \exists X_{p_1} \cdots \exists X_{p_s}\ \Psi_{\mathcal{T}} \wedge \Psi_{\mathcal{M}} \wedge \Psi_{\Gamma_0},$$

where the following hold.

—the unary predicates $X_{q_1}, \ldots, X_{q_n}, X_{\alpha_1}, \ldots, X_{\alpha_\ell}, X_{p_1}, \ldots, X_{p_s}$ are supposed to represent the states, the output alphabets of $\mathcal{T}$, and the states of $\mathcal{M}$, respectively;
—The formula $\Psi_{\mathcal{T}}$ expresses the behaviour of the transducer $\mathcal{T}$, that is, a tree satisfies $\Psi_{\mathcal{T}}$ in which for every node $u \in \mathsf{Dom}(t)$, $X_{q_i}(u)$ and $X_{\alpha_j}(u)$ holds, if there exists an accepting run of $\mathcal{T}$ on t in which the node u is labeled with q_i and output α_j;
—the formula $\Psi_{\mathcal{M}}$ expresses the behaviour of the automaton $\mathcal{M}$;
—the formula Ψ_{Γ_0} expresses the property that for every $\alpha_i \in \Gamma_0$, all the nodes belonging to X_{α_i} contain different data values, which is

$$\bigwedge_{\alpha \in \Gamma_0} \forall x \forall y (X_\alpha(x) \wedge X_\alpha(y) \wedge x \sim y \rightarrow x = y).$$

The construction of the formula $\Psi_{\mathcal{T}}$ is rather standard, thus, omitted. We will show the construction of the formula $\Psi_{\mathcal{M}}$. Let $\Phi_{[S]}(x)$ denote the following formula

$$\bigvee_{\alpha_i \in S} X_{\alpha_i}(x) \wedge \bigwedge_{\alpha_i \in S} \exists y (X_{\alpha_i}(x) \wedge x \sim y) \wedge \bigwedge_{\alpha_j \notin S} \forall y (X_{\alpha_j}(y) \rightarrow x \nsim y),$$

which states that the data value on the node x belongs to $[S]$. The formula $\Psi_{\mathcal{M}}$ expresses the following properties.

—That the node contains the minimal data value belongs to X_{p_1}. Formally, it can be written as follows.

$$\forall x (\forall y x \prec y \vee x \sim y \rightarrow X_{p_1}(x)).$$

—That the transition μ of $\mathcal{M}$ must be "respected." Formally, it can be written as follows.

$$\bigwedge_{(p_i, S, p_j) \in \mu} \Big(\forall x \forall y (X_{p_i}(x) \wedge \Psi_{[S]}(x) \wedge x \prec_{suc} y \rightarrow X_{p_j}(y)) \Big),$$

where $x \prec_{suc} y$ stands for $x \prec y \wedge \forall z (\neg (x \prec z \wedge z \prec y))$.

—That the node contains the maximal data value belongs to one of the final states of $\mathcal{M}$, denoted by F. Formally, it can be written as follows.

$$\forall x(\forall y\ (y \prec x \vee y \sim x) \rightarrow \bigvee_{p_i \in F} X_{p_i}(x)).$$

That the construction takes polynomial time is straightforward. This completes our proof of Theorem 5.8.

5.3. Proof of Theorem 5.9

The proof of Theorem 5.9 consists of two main steps.

(1) We prove that for each ODTA $\mathcal{S}$, if $\mathcal{L}_{data}(\mathcal{S}) \neq \emptyset$, then $\mathcal{L}_{data}(\mathcal{S})$ contains a data tree with *small model property* (Lemma 5.15).
(2) We describe a procedure, that given an ODTA $\mathcal{S}$, checks whether $\mathcal{L}(\mathcal{S})$ contains a data tree with small model property, by converting the ODTA $\mathcal{S}$ into an APC $(\mathcal{A}, \xi)$. Since the non-emptiness of APC is decidable, Theorem 5.9 follows immediately.

The first step (Lemma 5.15) is adapted from the proof of Bojanczyk et al. [2009, Proposition 3.10]. It is in the second step that our proof differs from theirs. The decision procedure in Bojanczyk et al. [2009] relies on intricate counting argument of the so called dog and sheep symbols (see [Bojanczyk et al. 2009, page 36]), and it seems that it cannot be generalised to the case of ODTA. On the other hand, our decision procedure relies mainly on Proposition 2.3, Lemma 4.1, and counting the cardinality of each $[S]$.

We need a few terminologies. A set of nodes in a data tree t is called connected if it is connected in the graph induced by $E_{\downarrow}$ and $E_{\rightarrow}$. A *zone* in a data tree t is a maximal connected set of nodes with the same data value. The *outdegree* of a zone Z is the number of different zones to which there is an edge (either $E_{\downarrow}$ or $E_{\rightarrow}$) from Z.

Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ be an ODTA, where $\mathcal{T}$ is a transducer from Σ to Γ. Let Q be the set of states of $\mathcal{T}$. For a tree $t \in \mathcal{L}_{data}(\mathcal{S})$, its extended tree $\tilde{t}$ (with respect to the ODTA $\mathcal{S}$) is a tree over the alphabet $\Sigma \times \{\top, \bot, *\}^3 \times Q \times \Gamma$, where

—the projection of $\tilde{t}$ to $\Sigma \times \{\top, \bot, *\}^3$ is $\mathsf{Profile}(t)$;
—the projection of $\tilde{t}$ to Q is an accepting run of $\mathcal{T}$ on t;
—the projection of $\tilde{t}$ to Γ is an output of $\mathcal{T}$ on t.

The following Lemma is simply an adaptation of Bojanczyk et al. [2009, Proposition 3.10] to the case of ODTA. The proof is via cut-and-paste, where given an ordered-data tree t over the alphabet Σ, where t has many zones in which the outdegree is large, we can cut some nodes in t and paste it in another part of t without affecting the set $V_t(a)$'s for each $a \in \Sigma$. The aim of such cut-and-paste is to reduce the number of zones in t with large outdegree. We give the following formal statement.

LEMMA 5.15 (COMPARE [BOJANCZYK ET AL. 2009, PROPOSITION 3.10]). *For every ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ over the alphabet Σ, if $\mathcal{L}_{data}(\mathcal{S}) \neq \emptyset$, then there exists a data tree $t \in \mathcal{L}_{data}(\mathcal{S})$ in which there are at most $K^{O(K^2)}$ zones with outdegree $\geq K^{(K^3)}$, where $K = O(|\Sigma| \cdot |Q| \cdot |\Gamma|)$ and Q is the set of states of $\mathcal{T}$ and Γ the output alphabet of $\mathcal{T}$.*

PROOF. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ be an ODTA over the alphabet Σ, and Q is the set of states of $\mathcal{T}$ and Γ the output alphabet of $\mathcal{T}$. Suppose that $t_0 \in \mathcal{L}_{data}(\mathcal{S})$. We will work on the extended tree $\tilde{t}_0$ of t_0. The aim is to convert $\tilde{t}_0$ into another tree $\tilde{t}$ over the alphabet $\Sigma \times \{\top, \bot, *\}^3 \times Q \times \Gamma$ such that

(1) the number of zones in $\tilde{t}$ with outdegree $\geq K^{(K^3)}$ is bounded by $K^{O(K^2)}$,
(2) the $\{\top, \bot, *\}^3$ projection of $\tilde{t}$ is the profile of each node,

(3) the Q projection of $\tilde{t}$ is an accepting run of $\mathcal{T}$ on the $\Sigma \times \{\top, \bot, *\}^3$ projection of $\tilde{t}$ and the output is its Γ projection,
(4) for each $(a, (l, p, r), q, b) \in \Sigma \times \{\top, \bot, *\}^3 \times Q \times \Gamma$, the set of data values found in the $(a, (l, p, r), q, b)$-nodes in $\tilde{t}_0$ is the same as the set of those found in $(a, (l, p, r), q, b)$-nodes in $\tilde{t}$,
(5) the Σ projection of $\tilde{t}$ is accepted by $\mathcal{S}$.

Intuitively, the tree $\tilde{t}$ is obtained via repeated applications of "pumping lemma" on both $E_{\downarrow}$- and $E_{\rightarrow}$-directions in the tree t.

Next, we give a brief summary of the proof adapted from the proof of Bojanczyk et al. [2009, Proposition 3.10]. We need the following terminologies, all of them are from Bojanczyk et al. [2009].

—Two nodes in a tree are called *siblings* if they have the same parent node.
—The set of all children of a node is called a *sibling group*.
—A contiguous sequence of siblings is called an *interval*.
We write $[u, v]$ for an interval in which u and v are the leftmost and rightmost nodes, respectively, in the interval.
—An interval $[u, v]$ is *complete* if the following hold.
—If a node u' exists such that $E_{\rightarrow}(u', u)$, then $u' \nsim u$.
—If a node v' exists such that $E_{\rightarrow}(v, v')$, then $u' \nsim u$.
—An interval is *pure* if all of its nodes have the same data value.
—A pure interval with the data value d is called a d-pure interval.
—If the parent of an interval (or, a sibling group) has data value d, then it is called a *d-parent interval* (or a *d-parent sibling group*).
—A zone with the data value d is called a *d-zone*.

The construction of $\tilde{t}$ from $\tilde{t}_0$ is as follows.

(1) Convert $\tilde{t}_0$ to another tree $\tilde{t}_1$ such that
—for every data value $d \in V_{\tilde{t}_1}$, there are at most $O(K)$ complete d-pure intervals of size more than $O(K)$;
—$V_{\tilde{t}_1}(a, (l, p, r), q, b) = V_{\tilde{t}_0}(a, (l, p, r), q, b)$, for every $(a, (l, p, r), q, b) \in \Sigma \times \{\top, \bot, *\}^3 \times Q \times \Gamma$;
—$\tilde{t}_1$ is an extended tree of its Σ projection with respect to $\mathcal{S}$.
This step is adapted from Bojanczyk et al. [2009, Proposition 3.12]. The idea is to cut an interval (together with its subtree) and paste it in another interval; and while doing so, the data values in the interval remain untouched.
(2) Convert $\tilde{t}_1$ to another tree $\tilde{t}_2$ such that
—for every data value $d \in V_{\tilde{t}_2}$, there are at most $O(K)$ d-parent sibling group with more than $K^{O(K)}$ complete pure intervals:
—$V_{\tilde{t}_2}(a, (l, p, r), q, b) = V_{\tilde{t}_1}(a, (l, p, r), q, b)$, for every $(a, (l, p, r), q, b) \in \Sigma \times \{\top, \bot, *\}^3 \times Q \times \Gamma$;
—$\tilde{t}_2$ is an extended tree of its Σ projection with respect to $\mathcal{S}$.
This step is adapted from Bojanczyk et al. [2009, Proposition 3.14]. Again when the cut-and-paste is performed the data values in the sibling groups remain untouched.
(3) Convert $\tilde{t}_2$ to another tree $\tilde{t}_3$ such that
—for every data value $d \in V_{\tilde{t}_3}$, there are at most $O(K)$ d-zones containing a path with more than $O(K)$ nodes;
—$V_{\tilde{t}_3}(a, (l, p, r), q, b) = V_{\tilde{t}_2}(a, (l, p, r), q, b)$, for every $(a, (l, p, r), q, b) \in \Sigma \times \{\top, \bot, *\}^3 \times Q \times \Gamma$;
—$\tilde{t}_3$ is an extended tree of its Σ projection with respect to $\mathcal{S}$.

This step is adapted from Bojanczyk et al. [2009, Proposition 3.17]. Again, when the cut-and-paste is performed, the data values in the zones remain untouched.

(4) Convert $\tilde{t}_3$ to another tree $\tilde{t}_4$ such that
—there are at most $K^{O(K^2)}$ complete pure intervals with more than $O(K^2)$ nodes;
—$V_{\tilde{t}_3}(a,(l,p,r),q,b) = V_{\tilde{t}_4}(a,(l,p,r),q,b)$, for every $(a,(l,p,r),q,b) \in \Sigma \times \{\top,\bot,*\}^3 \times Q \times \Gamma$;
—$\tilde{t}_4$ is an extended tree of its Σ projection with respect to $\mathcal{S}$.
This step is adapted from Bojanczyk et al. [2009, Proposition 3.20]. Here, actually, when the cut-and-paste is performed, the data values in some zones have to be changed. However, those changes are only applied to the *safe* zones, where a zone is safe if for every node in it there is another node outside the zone with the same label (from $\Sigma \times \{\top,\bot,*\} \times Q \times \Gamma$) and the same data value (see [Bojanczyk et al. 2009, page 23].) More specifically, these changes are done by applying Bojanczyk et al. [2009, Lemma 3.19] on the safe zones. That it is applied only on safe zones is important so that after changing the data values, constraints such as $\forall x \exists y(a(x) \to x \sim y \wedge b(y))$ are still satisfied.

(5) Convert $\tilde{t}_4$ to another tree $\tilde{t}_5$ such that
—there are at most $K^{O(K^2)}$ sibling groups containing more than $K^{O(K)}$ complete pure intervals;
—$V_{\tilde{t}_4}(a,(l,p,r),q,b) = V_{\tilde{t}_5}(a,(l,p,r),q,b)$, for every $(a,(l,p,r),q,b) \in \Sigma \times \{\top,\bot,*\}^3 \times Q \times \Gamma$;
—$\tilde{t}_5$ is an extended tree of its Σ projection with respect to $\mathcal{S}$.
This step is adapted from Bojanczyk et al. [2009, Proposition 3.21]. Here there are also changes of data values when performing cut-and-paste. However, as in the previous step, they are only applied to the *safe* zones. These changes are also done by applying Bojanczyk et al. [2009, Lemma 3.19] on the safe zones.

(6) Convert $\tilde{t}_5$ to another tree $\tilde{t}_6$ such that
—there are at most $K^{O(K^2)}$ zones containing paths with more than $O(K^2)$ nodes;
—$V_{\tilde{t}_5}(a,(l,p,r),q,b) = V_{\tilde{t}_6}(a,(l,p,r),q,b)$, for every $(a,(l,p,r),q,b) \in \Sigma \times \{\top,\bot,*\}^3 \times Q \times \Gamma$;
—$\tilde{t}_6$ is an extended tree of its Σ projection with respect to $\mathcal{S}$.
This step is adapted from Bojanczyk et al. [2009, Proposition 3.25]. Here there are also changes of data values when performing cut-and-paste. However, as in the previous step, they are only applied to the *safe* zones. More specifically, these changes are done by applying Bojanczyk et al. [2009, Lemma 3.24] on the safe zones.

The extended tree $\tilde{t}_6$ is the desired extended tree. It is a rather straightforward computation that there are at most $K^{O(K^2)}$ zones in $\tilde{t}_6$ with outdegree $\geq K^{(K^3)}$. □

To describe the decision procedure for Theorem 5.9, we need a few more additional terminologies. For a data tree t over the alphabet Γ and $S \subseteq \Gamma$, an S-zone is a zone in which the labels of the nodes are precisely S. We write $V_t^{zone}(S)$ to denote the set of data values found in S-zones in t. For $P \subseteq 2^\Gamma$,

$$[P]_t^{zone} = \bigcap_{S \in P} V_t^{zone}(S) \cap \bigcap_{R \notin P} \overline{V_t^{zone}(R)}.$$

Suppose $d_1 < \cdots < d_m$ are all the data values in t. The *zonal string representation* of the data values in t, denoted by $\mathcal{V}_\Gamma^{zone}(t)$, is the string $P_1 \cdots P_m$ over the alphabet 2^{2^Γ} such that for each $i \in \{1,\ldots,m\}$, $d_i \in [P_i]_t^{zone}$.

A *zonal* ODTA is $\mathcal{S}' = \langle \mathcal{T}, \mathcal{M}', \Gamma_0 \rangle$, where $\mathcal{T}$ and Γ_0 are as in the definition of ODTA, and $\mathcal{M}'$ is a finite state automaton over the alphabet 2^{2^Γ}. A data tree t is accepted by the zonal ODTA $\mathcal{S}'$, if the following holds.

—Profile(t) is accepted by $\mathcal{T}$, yielding an output tree t' over the alphabet Γ.
—The string $\mathcal{V}_\Gamma^{zone}(t')$ is accepted by $\mathcal{M}'$.
—For each $a \in \Gamma_0$, all the data values found in the a-nodes in t' are different.

PROPOSITION 5.16. *For every ODTA $\mathcal{S}$, one can construct in* EXPTIME *its equivalent zonal ODTA.*

PROOF. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ and $\mathcal{M} = \langle Q, q_0, \delta, F \rangle$. Its equivalent zonal ODTA is defined as $\mathcal{S}' = \langle \mathcal{T}, \mathcal{M}', \Gamma_0 \rangle$, where $\mathcal{M}' = \langle Q, q_0, \delta', F \rangle$ and $\delta' = \{(q, P, q') \in Q \times 2^{2^\Gamma} \times Q \mid \exists (q, S, q') \in \delta \text{ such that } \bigcup_{R \in P} R = S\}$. It is straightforward to show that $\mathcal{L}_{data}(\mathcal{S}') = \mathcal{L}_{data}(\mathcal{S})$.

Note that the only difference between $\mathcal{S}$ and $\mathcal{S}'$ is the transitions δ and δ' in $\mathcal{M}$ and $\mathcal{M}'$, respectively. The membership $(q, P, q') \in \delta'$ can be checked in polynomial time in the size of (q, P, q') and δ. Since there are exponentially many (q, P, q'), the exponential time upper bound holds immediately. This completes the proof of Proposition 5.16. □

Briefly, our decision procedure for Theorem 5.9 works as follows. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ be the given ODTA, where Σ is the input alphabet of $\mathcal{T}$, Γ the output alphabet, and Q the set of states of $\mathcal{T}$. Let $K = 27 \cdot |\Sigma| \cdot |Q| \cdot |\Gamma|$. The decision procedure constructs an APC $(\mathcal{A}, \xi)$ such that $\mathcal{S}$ accepts an ordered-data tree t in which there are at most $K^{O(K^2)}$ zones with outdegree $\geq K^{(K^3)}$ if and only if $(\mathcal{A}, \xi)$ accepts an extended tree of t with respect to $\mathcal{S}$.

Its precise description is given as follows.

(1) Compute $K = 27 \cdot |\Sigma| \cdot |Q| \cdot |\Gamma|$.
(2) Convert $\mathcal{S}$ into its zonal ODTA $\mathcal{S}' = \langle \mathcal{T}, \mathcal{M}', \Gamma_0 \rangle$.
(3) Guess the following items.
   (a) A set $\mathcal{P} \subseteq 2^{2^\Gamma}$.
   (b) For each $P \in \mathcal{P}$, guess an integer $M_P \leq 2 \cdot K^{K^3} \cdot 2^K + 2 \cdot K^{K^3} + 1$ and a set of M_P constants $\mathcal{C}_P = \{c_1, \ldots, c_{M_P}\}$.[5]
   (c) Two integers N, N' such that $N' \leq N \leq K^{O(K^2)}$ and a set of N' constants $\mathcal{D} = \{d_1, \ldots, d_{N'}\}$.
   The intuitive meaning of N' and N are the number of zones with outdegree $\geq K^{(K^3)}$ and the number of data values found in them, respectively. We also remark that the constants in $\mathcal{D}$ may overlap with the constants in some $\mathcal{C}_P$.
   (d) For each $d \in \mathcal{D}$, guess a set $P_d \subseteq 2^\Gamma$.
(4) Construct the following automaton $\mathcal{A}$ over the alphabet $\Sigma \times \{\top, \bot, *\}^3 \times Q \times \Gamma$.
   (a) $\mathcal{A}$ accepts only the extended trees of $\mathcal{L}(\mathcal{T})$ in which there are at most N zones with outdegree $\geq K^{(K^3)}$.
   (b) The automaton $\mathcal{A}$ can remember the constants in its states.

[5]The purpose of the number $2 \cdot K^{K^3} \cdot 2^K + 2 \cdot K^{K^3}$ is the application of Lemma 4.1 later on, where we consider the graph where the nodes are the zones. Each zone is labeled with a symbol from $2^{\Sigma \times \{\top, \bot, *\}^3 \times Q \times \Gamma}$, which is of size 2^K. If a zone has outdegree $\leq K^{(K^3)}$, then it has only at most $K^{(K^3)}$ nodes, which means that its degree (the sum of indegree and outdegree) is bounded by $2 \cdot K^{K^3}$. Now $\mathcal{P}$ is intended to contain all those P's in which $|[P]_t^{zone}| \leq 2 \cdot K^{K^3} \cdot 2^K + 2 \cdot K^{K^3} + 1$ so that we can "guess" some constants as elements of $[P]_t^{zone}$ and make sure by automaton that the same constant is not "assigned" to adjacent zones. For P not in $\mathcal{P}$, we can apply Lemma 4.1 to make sure the same data value from $[P]_t^{zone}$ is not assigned to adjacent zones.

(c) For every $P \in \mathcal{P}$, for every $c \in \mathcal{C}_P$, the automaton $\mathcal{A}'$ assigns the constant c in an S-zone, for every $S \in P$, but not in any R-zone, for every $R \notin P$.
(d) The automaton $\mathcal{A}$ assigns every zone with outdegree $\geq K^{(K^3)}$ with a constant from $\mathcal{D}$.
(e) For every $d \in \mathcal{D}$, for every $S \in P_d$, the automaton $\mathcal{A}$ assigns the constant d in an S-zone, for every $S \in P_d$, but in no R-zone, for every $R \notin P_d$.
(f) For each $a \in \Gamma_0$, there is at most one a-node in every zone, and for every two zones that contains a-nodes, if they are assigned with some constants from $\mathcal{C}_P$'s and $\mathcal{D}$, then these constants must be different.
(g) For every two adjacent zones, if they are assigned with constants from $\mathcal{C}_P$'s and $\mathcal{D}$, then these constants must be different.

The automaton $\mathcal{A}$ assigns a constant to a zone by remembering the constant in the state when $\mathcal{A}$ is reading the zone.

(5) Let $P_1, \ldots, P_m$ be the enumeration of nonempty subsets of 2^Γ.
Applying Lemma 2.3, convert the automaton $\mathcal{M}'$ into its Presburger formula $\xi_{\mathcal{M}'}(z_{P_1}, \ldots, z_{P_m})$, where the intended meaning of z_{P_i}'s is the number of appearances of the label P_i.

(6) Let $\Gamma = \{a_1, \ldots, a_\ell\}$ and $S_1, \ldots, S_k$ be the enumeration of nonempty subsets of Γ. Define the formula $\xi(x_{a_1}, \ldots, x_{a_\ell}, x_{S_1}, \ldots, x_{S_k})$:

$$\exists z_{P_1} \cdots \exists z_{P_m} \ \xi_{\mathcal{M}'}(z_{P_1}, \ldots, z_{P_m}), \tag{2}$$

$$\wedge \bigwedge_{P_i \in \mathcal{P}} z_{P_i} = M_{P_i}, \tag{3}$$

$$\wedge \bigwedge_{P_i \notin \mathcal{P}} z_{P_i} \geq 2 \cdot K^{K^3} \cdot 2^K + 2 \cdot K^{(K^3)} + 1, \tag{4}$$

$$\wedge \bigwedge_{S \subseteq \Gamma} \left(x_S \ \geq \sum_{P_i \ni S \text{ and } P_i \notin \mathcal{P}} z_{P_i} \right), \tag{5}$$

$$\wedge \bigwedge_{a \in \Gamma_0} \left(x_a \ = \sum_{\substack{\text{there exists } S \text{ such that} \\ a \in S \text{ and } S \in P_i \text{ and } P_i \notin \mathcal{P}}} z_{P_i} \right), \tag{6}$$

$$\wedge \bigwedge_{P_i \notin \mathcal{P}} z_{P_i} \geq |\{d \in \mathcal{D} \mid P_d = P_i\}|. \tag{7}$$

The meaning of x_a is the number of a-nodes occurring in the zone not assigned with any constants from $\mathcal{C}_P$'s and $\mathcal{D}$; and x_S is the number S-zones not assigned with any constants from $\mathcal{C}_P$'s and $\mathcal{D}$. The intuition behind items (2)–(6) is rather clear. The intuition behind item (7) is as follows. Recall that in Step (3), for each $d \in \mathcal{D}$, we guess a set P_d. The meaning is that $d \in [P_d]_t^{zone}$ for some $t \in \mathcal{L}^{data}(\mathcal{S})$. So for every $P_i \notin \mathcal{P}$, the number of d such that $P_d = P_i$ should not exceed z_{P_i}. This is precisely what is stated in item (7).

(7) Test the nonemptiness of the APC $(\mathcal{A}, \xi)$.

Before we proceed to prove its correctness, we first present the analysis of its complexity.

—Step (1) is trivial and Step (2) takes exponential time.
—Step (3) takes nondeterministic exponential time in the size of $\mathcal{S}$. The analysis is as follows. Step (3a) takes nondeterministic exponential time in the size of 2^Γ, which

is bounded by the size of $\mathcal{M}$ in $\mathcal{S}$. (Recall that the alphabet in $\mathcal{M}$ is 2^Γ.) Step (3b) can guess up exponentially many constant in each $\mathcal{C}_P$, and there are exponentially many different $\mathcal{C}_P$, hence it takes double exponential time in the size of 2^Γ. Steps (3c) and (3d) take nondeterministic exponential time.

—Step (4) takes deterministic triple exponential time and can produce the automaton $\mathcal{A}$ of size up to triple exponential. The analysis is as follows. The automaton $\mathcal{A}$ has to remember in its states the outdegree of each zone up to $K^{(K^3)}$ and the number of zones with out degree $\geq K^{(K^3)}$. This induces an exponential blow-up in the size of $\mathcal{T}$. The number of constants in guessed in Step (3) is double exponential in the size of $\mathcal{T}$. Then $\mathcal{A}$ has to remember in its states which constant is assigned to which zone (of outdegree $\geq K^{(K^3)}$), which induces another exponential blow-up. Altogether the size of $\mathcal{A}$ can be triple exponential in the size of $\mathcal{T}$.

—By Proposition 2.3, Step (5) takes polynomial time in the size $\mathcal{M}'$, which is of size exponential in the size of the original $\mathcal{M}$.

—The length of the formula in Step (6) is double exponential in the size of $\mathcal{S}$, since the number of constants in $\mathcal{D}$ can be double exponential in the size of 2^Γ, and hence $\mathcal{S}$.

—Step (7) takes nondeterministic polynomial time in the size of $(\mathcal{A}, \xi)$, and hence nondeterministic triple exponential time in the size of the input $\mathcal{S}$.

The following claim immediately implies the correctness of our algorithm.

CLAIM 2. (1) *For every ordered-data tree $t \in \mathcal{L}_{data}(\mathcal{S})$, in which there are at most $K^{O(K^2)}$ zones with outdegree $\geq K^{(K^3)}$, there exists an extended tree of t which is accepted by the APC $(\mathcal{A}, \xi)$.*

(2) *For every $t' \in \mathcal{L}(\mathcal{A}, \xi)$, there exists an ordered-data tree $t \in \mathcal{L}_{data}(\mathcal{S})$ such that t' is an extended tree of t with respect to $\mathcal{S}$.*

PROOF. We prove (1) first. Let $t \in \mathcal{L}_{data}(\mathcal{S})$ be an ordered-data tree in which there are at most $K^{O(K^2)}$ zones with outdegree $\geq K^{(K^3)}$. Let t_0 be the output of $\mathcal{T}$ on t so that $\mathcal{V}^{zone}(t_0)$ is accepted by $\mathcal{M}$ and all nodes in t_0 labeled with a symbol in Γ_0 have different data values.

We have the following items guessed in Step 3 in our preceding algorithm.

—$\mathcal{P} = \{P \mid |[P]_t^{zone}| \leq 2 \cdot K^{K^3} \cdot 2^K + 2 \cdot K^{(K^3)} + 1\}$.
—For each $P \in \mathcal{P}$, $\mathcal{C}_P = [P]_{t_0}^{zone}$, and $M_P = |\mathcal{C}_P|$.
—N be the number of zones in t with outdegree $\geq K^{(O(K^2))}$ and N' be the number of data values found in these zones.
—$\mathcal{D} = \{d \mid d$ is found in a zone with outdegree $\geq K^{(K^3)}\}$.
—For each $d \in \mathcal{D}$, P_d is the set such that $d \in [P_d]_{t_0}^{zone}$.

Now let t' be an extended tree of t with respect to $\mathcal{S}$, and $\mathcal{A}$ and ξ be the automaton and formula as constructed in Steps (4)–(6). We are going to show that $t' \in \mathcal{L}(\mathcal{A}, \xi)$. Obviously, $t' \in \mathcal{L}(\mathcal{A})$. To show that the formula ξ is satisfied, we take $\mathsf{Parikh}(\mathcal{V}^{zone}(t_0))$ as witness to $(z_{P_1}, \ldots, z_{P_m})$. Since $\mathcal{V}^{zone}(t_0) \in \mathcal{L}(\mathcal{M}')$, by Proposition 2.3, the formula $\xi_{\mathcal{M}'}(\mathsf{Parikh}(\mathcal{V}^{zone}(t_0)))$ holds. It is straightforward from the definitions of the items $\mathcal{P}$, M_P's, N, N', $\mathcal{D}$, and P_d's that the formula ξ in Step (6) is satisfied with x_a's and x_S's interpreted as intended.

Now we prove (2). The proof is more delicate than the proof of (1). Suppose $t' \in \mathcal{L}(\mathcal{A}', \xi)$. We are going to construct an ordered-data tree t from t' such that t' is an extended tree of t with respect to $\mathcal{S}$. Let $\mathcal{P}$, M_P's, $\mathcal{C}_P$'s, N, N', $\mathcal{D}$, and P_d's the items as guessed in Step (3) and the following.

—For each $a_i \in \Gamma$, let n_{a_i} be the number of a_i-nodes in t' occurring in a zone without any constants from $\mathcal{C}_P$'s and $\mathcal{D}$.
—For each $S_i \subseteq \Gamma$, let n_{S_i} be the number of S_i-zones in t' without any constants from $\mathcal{C}_P$'s and $\mathcal{D}$.

Suppose $(k_{P_1}, \ldots, k_{P_m})$ be the witness to $z_{P_1}, \ldots, z_{P_m}$ such that

$$\xi(n_{a_1}, \ldots, n_{a_\ell}, n_{S_1}, \ldots, n_{S_l}) \qquad \text{holds.}$$

By Proposition 2.3, this means that there exists a word $w \in \mathcal{L}(\mathcal{M}')$ such that $\mathsf{Parikh}(w) = (k_{P_1}, \ldots, k_{P_m})$. For each P_i, we let

$$\mathcal{N}_{P_i} = \{j \mid \text{position } j \text{ in } w \text{ is labeled } P_i\}.$$

We will assign a data value to each node in t such that

$$[P_i]_t^{zone} = \mathcal{N}_{P_i},$$

and $\mathcal{V}^{zone}(t) = w$. The assignment is done according to the three following cases.

Case 1. For the nodes that are assigned with some constants from $\mathcal{C}_{P_i}$'s.

In this case, $P_i \in \mathcal{P}$. We define bijections $f_{P_i} : \mathcal{C}_{P_i} \mapsto \mathcal{N}_{P_i}$. There is always a bijection from $\mathcal{C}_{P_i}$ to $\mathcal{N}_{P_i}$ since they have the same cardinality M_{P_i}, due to the following condition in the formula ξ.

$$\bigwedge_{P_i \in \mathcal{P}} z_{P_i} = M_{P_i}.$$

The data value assignment to nodes of this case can be done by replacing every constant $c \in \mathcal{C}_{P_i}$ with $f_{P_i}(c)$.

Case 2. For the nodes that are assigned some constants from $\mathcal{D}$.

We define a 1-1 mapping $f : \mathcal{D} \mapsto \{1, \ldots, |w|\}$ such that $f(d) \in \mathcal{N}_{P_d}$, where P_d is the set guessed in Step 3. Such 1-1 mapping exists because the following condition in the formula ξ:

$$\bigwedge_{P_i \notin \mathcal{P}} z_{P_i} \geq |\{d' \in \mathcal{D} \mid P_{d'} = P_i\}|.$$

The data value assignment to nodes of this case can be done by replacing every constant $d \in \mathcal{D}$ with $f(d)$.

Case 3. For the nodes that are not assigned any constants from $\mathcal{C}_P$'s and $\mathcal{D}$.

First we assign each of such zone in t with a data value[6] such that for each $S \subseteq \Gamma$,

$$V_t^{zone}(S) = \bigcup_{P_i \ni S \text{ and } P_i \notin \mathcal{P}} \mathcal{N}_{P_i}.$$

This step can be done as follows. The number of such S-zone in t is greater than $\sum_{P_i \ni S \text{ and } P_i \notin \mathcal{P}} |\mathcal{N}_{P_i}|$, due to the following condition in the formula ξ:

$$x_S \geq \sum_{P_i \ni S \text{ and } P_i \notin \mathcal{P}} z_{P_i}.$$

Thus, we can simply assign every S-zone with a data value from $\bigcup_{P_i \ni S \text{ and } P_i \notin \mathcal{P}} \mathcal{N}_{P_i}$ and make sure every data value from $\bigcup_{P_i \ni S \text{ and } P_i \notin \mathcal{P}} \mathcal{N}_{P_i}$ appears in some S-zone.

[6]A zone in t can be recognised from the profile information in t'.

However, by assigning data values like that, some adjacent zones may get the same data values. Here we apply Lemma 4.1. Since for each $P_i \notin \mathcal{P}$, $|\mathcal{N}_{P_i}| \geq 2 \cdot K^{K^3} \cdot 2^K + 2 \cdot K^{(K^3)} + 1$, by the following condition in the formula ξ

$$\bigwedge_{P_i \notin \mathcal{P}} z_{P_i} \geq 2 \cdot K^{K^3} \cdot 2^K + 2 \cdot K^{(K^3)} + 1,$$

the cardinality

$$\Big| \bigcup_{P_i \ni S \text{ and } P_i \notin \mathcal{P}} \mathcal{N}_{P_i} \Big| = \sum_{P_i \ni S \text{ and } P_i \notin \mathcal{P}} |\mathcal{N}_{P_i}| \geq 2 \cdot K^{K^3} \cdot 2^K + 2 \cdot K^{(K^3)} + 1.$$

The outdegree of such zone is $\leq K^{(K^3)}$, hence, the number of nodes in the zone is also $\leq K^{(K^3)}$. Since each node can have indegree at most 1, the degree of each of such zone is $\leq 2 \cdot K^{(K^3)}$. By applying Lemma 4.1, where $\deg(G) = 2 \cdot K^{(K^3)}$, we can reassign the data value in such zone so that each adjacent zone get different data value.

This completes the proof of our claim. □

6. WEAK ODTA

A weak ODTA over Σ is a triplet $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$, where $\mathcal{T}$ is a letter-to-letter transducer from Σ to the output alphabet Γ, and $\mathcal{M}$ is a finite state automaton over 2^Γ and $\Gamma_0 \subseteq \Gamma$. An ordered-data tree t is accepted by $\mathcal{S}$, denoted by $t \in \mathcal{L}_{data}(\mathcal{S})$, if there exists an ordered-data tree t' over Γ such that

—on input $\mathsf{Proj}(t)$, the transducer $\mathcal{T}$ outputs t';
—the automaton $\mathcal{M}$ accepts the string $\mathcal{V}_\Gamma(t')$;
—for every $a \in \Gamma_0$, all the a-nodes in t' have different data values.

Note that the only difference between weak ODTA and ODTA is the equality test on the data values in neighboring nodes. Such difference is the cause of the triple exponential leap in complexity, as stated in the following theorem.

THEOREM 6.1. *The non-emptiness problem for weak ODTA is in* NP.

PROOF. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ be a weak ODTA. Let Σ, Q, Γ be the input alphabet, set of states and output alphabet of $\mathcal{T}$, respectively.

We need the following notation. For a tree $t \in \mathcal{L}_{data}(\mathcal{S})$, its extended tree $\tilde{t}$ (with respect to the weak ODTA $\mathcal{S}$) is a tree over the alphabet $\Sigma \times Q \times \Gamma$, where

—the projection of $\tilde{t}$ to Σ is t;
—the projection of $\tilde{t}$ to Q is an accepting run of $\mathcal{T}$ on t such that its output is the projection of $\tilde{t}$ to Γ.

The decision procedure for Theorem 6.1 works as follows.

(1) Construct an automaton $\mathcal{A}$ over the alphabet $\Sigma \times Q \times \Gamma$ for the extended trees accepted by $\mathcal{T}$.
(2) Let $\mathcal{P} = \{S_1, \ldots, S_m\} \subseteq 2^\Gamma$ be the set of symbols used in $\mathcal{M}$.
By applying Proposition 2.3, construct the Presburger formula $\xi_{\mathcal{M}}(x_{S_1}, \ldots, x_{S_m})$ for $\mathcal{M}$.

(3) Let $\Sigma \times Q \times \Gamma = \{(a_1, q_1, \alpha_1), \ldots, (a_k, q_n, \alpha_\ell)\}$. Let $\varphi(x_{(a_1,q_1,\alpha_1)}, \ldots, x_{(a_k,q_n,\alpha_\ell)})$ be the following formula.

$$\exists x_{\alpha_1} \cdots \exists x_{\alpha_\ell}\ \exists x_{S_1} \cdots \exists x_{S_m}\ \xi_{\mathcal{M}}(x_{S_1}, \ldots, x_{S_m})$$
$$\wedge \bigwedge_{\alpha_i \in \Gamma} \left(x_{\alpha_i} = \sum_{a_j \in \Sigma, q_h \in Q} x_{(a_j,q_h,\alpha_i)} \right)$$
$$\wedge \bigwedge_{\alpha_i \in \Gamma} \left(x_{\alpha_i} \geq \sum_{\alpha_i \in S_j} x_{S_j} \right) \wedge \bigwedge_{\alpha_i \in \Gamma_0} \left(x_{\alpha_i} = \sum_{\alpha_i \in S_j} x_{S_j} \right).$$

(4) Test the non-emptiness of APC $(\mathcal{A}, \varphi(x_{(a_1,q_1,\alpha_1)}, \ldots, x_{(a_k,q_n,\alpha_\ell)}))$.

That this procedure works in NP follows directly from the fact that the non-emptiness problem of APC is in NP.

We now show the correctness of our algorithm by showing that $\mathcal{L}_{data}(\mathcal{S}) \neq \emptyset$ if and only if $\mathcal{L}(\mathcal{A}, \varphi) \neq \emptyset$. (For the sake of presentation, we write φ without its free variables.) We start with the "only if" part. Suppose that $t \in \mathcal{L}_{data}(\mathcal{S})$. We claim that the extended tree $\tilde{t}$ of t is accepted by $(\mathcal{A}, \varphi)$. Obviously, $\tilde{t} \in \mathcal{L}(\mathcal{A})$. To show that $\varphi(\mathsf{Parikh}(\tilde{t}))$ holds, let t' be the data tree obtained by projecting $\tilde{t}$ to Γ and the data value in each node comes from the same node in t. That is, t' is an output of $\mathcal{T}$ on t. We will show that $\varphi(\mathsf{Parikh}(\tilde{t}))$ holds.

—As witness to $x_{S_1}, \ldots, x_{S_m}$, we take $\mathsf{Parikh}(\mathcal{V}(t'))$. Since $\mathcal{V}(t') \in \mathcal{L}(\mathcal{M})$, by Proposition 2.3, $\xi_{\mathcal{M}}(\mathsf{Parikh}(\mathcal{V}(t')))$ holds.

—As witness to $x_{\alpha_1}, \ldots, x_{\alpha_\ell}$, we take $\mathsf{Parikh}(t')$. Now for each $\alpha_i \in \Gamma$, the constraint $x_{\alpha_i} \geq \sum_{\alpha_i \in S_j} x_{S_j}$ holds since the number of data values in the α_i-nodes cannot exceed the number of α_i-nodes itself. The constraint $x_{\alpha_i} = \sum_{\alpha_i \in S_j} x_{S_j}$, for each $\alpha_i \in \Gamma_0$, since the data values found in α_i-nodes are all different.

Thus, $\varphi(\mathsf{Parikh}(\tilde{t}))$ holds, and this concludes our proof of the "only if" part.

Now we prove the "if" part. Suppose that $\tilde{t} \in \mathcal{L}(\mathcal{A}, \varphi)$. So $\tilde{t} \in \mathcal{L}(\mathcal{A})$. Let t and t' be the Σ- and Γ-projection of $\tilde{t}$, respectively. By the definition of $\mathcal{A}$, t' is an output of $\mathcal{T}$ on t. Now since $\varphi(\mathsf{Parikh}(\tilde{t}))$ holds, in particular, there exists a witness $\bar{M} = (M_1, \ldots, M_m)$ to $x_{S_1}, \ldots, x_{S_m}$ such that $\xi_{\mathcal{M}}(\bar{M})$ holds, by Proposition 2.3, there exists a word $w \in \mathcal{L}(\mathcal{M})$ over the alphabet 2^Γ such that $\mathsf{Parikh}(w) = \bar{M}$.

We are going to assign data values to the nodes of t' (thus, also to those of t) such that $t \in \mathcal{L}_{data}(\mathcal{S})$. The assignment is done as follows. For each $S \subseteq \Gamma$, let $V_w(S)$ be the set of positions of w labeled with S. Now for each $\alpha \in \Gamma$, we assign the α-nodes in t' with the data values from $\bigcup_{\alpha \in S} V_w(S)$ such that $V_{t'}(\alpha) = \bigcup_{\alpha \in S} V_w(S)$. This is possible due to the constraint $x_\alpha \geq \sum_{\alpha \in S} x_S$.

With such assignment, we get $\mathcal{V}(t') = w$. Thus, $\mathcal{V}(t') \in \mathcal{L}(\mathcal{M})$. Moreover, for every $\alpha \in \Gamma_0$, all the data values in α-nodes are different, which follows from the constraint $x_\alpha = \sum_{\alpha \in S} x_S$. Therefore, the resulting ordered-data tree $t \in \mathcal{L}_{data}(\mathcal{S})$. This concludes our proof. □

Next, we give the logical characterisation of weak ODTA.

THEOREM 6.2. *A language $\mathcal{L}$ is accepted by a weak ODTA if and only if $\mathcal{L}$ is expressible with a formula of the form: $\exists X_1 \cdots \exists X_m\, \varphi \wedge \psi$, where φ is a formula from $FO^2(E_\downarrow, E_\rightarrow)$, and ψ is a formula from $FO(\sim, \prec, \prec_{suc})$, extended with the unary predicates $X_1, \ldots, X_m$.*

The proof of Theorem 6.2 is the same as the proof of Theorem 5.8. The difference is that to simulate the $\mathrm{FO}^2(E_\downarrow, E_\rightarrow)$ formula φ, the profile information is not necessary. The complexity of the translation is still the same as in Theorem 5.8.

6.1. Extending Weak ODTA with Presburger Constraints

Like in the case of APC, we can extend weak ODTA with Presburger constraints without increasing the complexity of its non-emptiness problem. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ be a weak ODTA, where Σ and Γ are the input and output alphabets of $\mathcal{T}$, respectively. Let $\Gamma = \{\alpha_1, \ldots, \alpha_\ell\}$.

A weak ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ extended with Presburger constraint is a tuple $\langle \mathcal{S}, \xi \rangle$, where $\xi(x_1, \ldots, x_\ell, y_1, \ldots, y_{2^\ell-1})$ is an existential Presburger formula with the free variables $x_1, \ldots, x_\ell, y_1, \ldots, y_{2^\ell-1}$. An ordered-data tree t is accepted by $\langle \mathcal{S}, \xi \rangle$, if there exists an output t' of $\mathcal{T}$ on t, the automaton $\mathcal{M}$ accepts $\mathcal{V}_\Gamma(t')$, for each $a \in \Gamma_0$, all a-nodes in t' have different data values and $\xi(\mathsf{Parikh}(t'), \mathsf{Parikh}(\mathcal{V}_\Gamma(t')))$ holds. We write $\mathcal{L}_{data}(\mathcal{S}, \xi)$ to denote the set of languages accepted by $\langle \mathcal{S}, \xi \rangle$.

We claim that the non-emptiness problem of weak ODTA extended with Presburger constraint is still decidable in NP. The reason is as follows. The non-emptiness of a weak ODTA $\mathcal{S}$ is checked by converting $\mathcal{S}$ into an APC $(\mathcal{A}, \varphi)$, where φ expresses linear constraints on the number of nodes labeled with symbols from Σ and Γ as well as those labeled with Q in the accepting run. The formula ξ can be appropriately inserted into φ, and hence, the non-emptiness of $(\mathcal{S}, \xi)$ is reducible to non-emptiness of APC, which is in NP.

6.2. Comparison with Other Known Decidable Formalisms

We are going to compare the expressiveness of weak ODTA with other known models with decidable non-emptiness.

6.2.1. DTD with Integrity Constraints. An XML document is typically viewed as a data tree. The most common XML formalism is document type definition (DTD). In short, a DTD is a contextfree grammar, and a tree t conforms to a DTD D if it is a derivation tree of a word accepted by the context free grammar.

The most commonly used XML constraints are integrity constraints which are of two types.

—The *key constraint* $key(a)$ is the following constraint:

$$\forall x \forall y (a(x) \wedge a(y) \wedge x \sim y \rightarrow x = y).$$

—The *inclusion constraint* $V(a) \subseteq V(b)$ is the following constraint:

$$\forall x \exists y (a(x) \rightarrow b(y) \wedge x \sim y).$$

The satisfiability problem of a given DTD D and a collection $\mathcal{C}$ of integrity constraints asks whether there exists an ordered-data tree t that conforms to the DTD that satisfies all the constraints in $\mathcal{C}$. Fan and Libkin [2002] show that this problem is NP-complete.

THEOREM 6.3. *Given a DTD D and a collection $\mathcal{C}$ of integrity constraints, one can construct a weak ODTA $\mathcal{S}$ such that $\mathcal{L}_{data}(\mathcal{S})$ is precisely the set of ordered-data trees that conforms to D and satisfies all constraints in $\mathcal{C}$.*

PROOF. Let Σ be the alphabet of the given DTD D. Consider the following weak ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Sigma_0 \rangle$.

—$\mathcal{T}$ is an identity transducer that checks whether the input tree conforms to DTD D.
—$\mathcal{M}$ is an automaton that accepts $\mathcal{P}^*$, where $\mathcal{P} = 2^\Sigma - \{S \mid a \in S \text{ and } b \notin S \text{ for some } V(a) \subseteq V(b) \in \mathcal{C}\}$.
—$\Sigma_0 = \{a \mid key(a) \in \mathcal{C}\}$.

That $\mathcal{S}$ is the desired ODTA follows immediately from the fact that for every ordered-data tree t, $V_t(a) \subseteq V_t(b)$ if and only if $[S]_t = \emptyset$ for all S where $a \in S$, but $b \notin S$. □

The size of the automaton $\mathcal{M}$, hence the size of $\mathcal{S}$, produced by our construction in Theorem 6.3 is of exponential size. This blow-up is tight, as the following example shows. Consider the case where $\mathcal{C}$ does not contain inclusion constraints. That is, $\mathcal{C}$ contains only key constraints. Then any equivalent ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Sigma_0 \rangle$ will have $\mathcal{L}(\mathcal{M}) = (2^\Sigma - \{\emptyset\})^*$. Thus, we have exponential blow-up in the size of $\mathcal{M}$. Nevertheless, if we are concerned only with satisfiability, then we can lower the complexity to NP as stated in the following theorem.

THEOREM 6.4. *Given a DTD D and a collection $\mathcal{C}$ of integrity constraints, one can construct a weak ODTA $\mathcal{S}$ in nondeterministic polynomial time such that $\mathcal{L}_{data}(\mathcal{S}) \neq \emptyset$ if and only if there exists an ordered-data tree t that conforms to D and satisfies all the constraints in $\mathcal{C}$.*

PROOF. Let Σ be the alphabet of the DTD D. We nondeterministically construct a weak ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Sigma_0 \rangle$ as follows.

—$\mathcal{T}$ is an identity transducer that checks whether the input tree conforms to DTD D.
—Guess a sequence $(H_1, \ldots, H_k)$ of some subsets of Σ such that
  —Σ is partitioned into $H_1 \cup \cdots \cup H_k$.
  —For every two different symbols $a, b \in \Sigma$, a, b are in the same set H_i if and only if both $V(a) \subseteq V(b)$ and $V(b) \subseteq V(a)$ are in $\mathcal{C}$;
  —If $V(a) \subseteq V(b) \in \mathcal{C}$ and $V(b) \subseteq V(a) \notin \mathcal{C}$, then $a \in H_i$ and $b \in H_j$ and $i \leq j$.
  Intuitively, the sequence $(H_1, \ldots, H_k)$ tells us the ordering of the elements in Σ that respect the inclusion constraints in $\mathcal{C}$, where if both $V(a) \subseteq V(b)$ and $V(b) \subseteq V(a)$ are in $\mathcal{C}$, then a and b are tie and they must be in the same set H_i.
—Let $S_1, \ldots, S_k \subseteq \Sigma$ be such that $S_i = \Sigma - (H_1 \cup \cdots \cup H_{i-1})$, where $S_1 = \Sigma$ and $S_k = H_k$.
—$\mathcal{M}$ is a nondeterministic automaton over the alphabet $\{S_1, \ldots, S_k\}$, where the set of states is $\{q_1, \ldots, q_k\}$, all $q_1, \ldots, q_k$ are the initial states and the final states, and the transitions are: (q_i, S_j, q_j) for every $1 \leq i \leq j \leq k$.
—$\Sigma_0 = \{a \mid key(a) \in \mathcal{C}\}$.

We claim that $\mathcal{L}_{data}(\mathcal{S}) \neq \emptyset$ if and only if there exists an ordered-data tree t that conforms to D and satisfies all the constraints in $\mathcal{C}$.

We start with the "if" direction. Suppose t conforms to the DTD D and satisfies all the constraints in $\mathcal{C}$. For each $a \in \Sigma$, let N_a be the number of data values found in the a-nodes in t. Let $(H_1, \ldots, H_k)$ be a sequence of some subsets of Σ such that

—Σ is partitioned into $H_1 \cup \cdots \cup H_k$;
—For every two different symbols $a, b \in \Sigma$, a, b are in the same set H_i if and only if $N_a = N_b$;
—$a \in H_i$ and $b \in H_j$ and $i \leq j$ if and only if $N_a \leq N_b$.

Consider the following ordered-data tree t' over Σ, where t' is obtained from t by reassigning the data values on the nodes in t as follows. For each $a \in \Sigma$, we assign the set of integers $\{d \mid 1 \leq d \leq N_a\}$ as the data values of a-nodes in t'. Such assignment is possible since N_a is no more than the number of a-nodes in t'. With such assignment t' still obeys the constraints in $\mathcal{C}$, as shown next.

—If $key(a) \in \mathcal{C}$, then N_a is precisely the number of a-nodes in t, thus, also in t'. Thus, with the data values $\{1, \ldots, N_a\}$, the data values on the a-nodes in t' are all different.
—If $V(a) \subseteq V(a') \in \mathcal{C}$, then obviously, $N_a \leq N_{a'}$. Thus, t' still satisfies the constraint $V(a) \subseteq V(a')$, since the data values in a-nodes in t' are $\{1, 2, \ldots, N_a\}$, while those in a'-nodes are $\{1, 2, \ldots, N_{a'}\}$.

Now the string $\mathcal{V}(t')$ is of the form $R_1 \cdots R_m$, where $m = \max_{a \in \Sigma}(N_a)$, where $R_1 \supseteq R_2 \supseteq \cdots \supseteq R_m$, and if $R_i \neq R_{i+1}$, then $R_{i+1} - R_i = H_j$ for some H_j in the sequence $(H_1, \ldots, H_k)$.

By the definition of $\mathcal{M}$, $\mathcal{V}(t')$ is accepted by $\mathcal{M}$. That t is accepted by $\mathcal{T}$ is trivial and so is the fact that all the data values found in a-nodes in t' for each $a \in \Sigma_0$. Thus, $t' \in \mathcal{L}_{data}(\mathcal{S})$.

For the "only if" direction, it is sufficient to observe that for every sequence $(H_1, \ldots, H_k)$ that "respects" the inclusion constraints in $\mathcal{C}$, as previously explained, if $\mathcal{V}(t) \in \mathcal{L}(\mathcal{M})$, then t satisfies all the inclusion constraints in $\mathcal{C}$. This completes our proof. □

6.2.2. Set and Linear Constraints for Data Trees. In David et al. [2012] the *set and linear constraints* are introduced for data trees. As argued there, those constraints, together with automata, are able to capture many interesting properties commonly used in XML practice. We review those constraints and show how they can be captured by weak ODTA extended with Presburger constraints.

Data terms (or just terms) are given by the grammar

$$\tau := V(a) \mid \tau \cup \tau \mid \tau \cap \tau \mid \overline{\tau} \qquad \text{for } a \in \Sigma.$$

The semantics of τ is defined with respect to a data tree t.

$$\llbracket V(a) \rrbracket_t = V_t(a) \qquad \llbracket \overline{\tau} \rrbracket_t = V_t - \llbracket \tau \rrbracket_t$$
$$\llbracket \tau_1 \cap \tau_2 \rrbracket_t = \llbracket \tau_1 \rrbracket_t \cap \llbracket \tau_2 \rrbracket_t. \qquad \llbracket \tau_1 \cup \tau_2 \rrbracket_t = \llbracket \tau_1 \rrbracket_t \cup \llbracket \tau_2 \rrbracket_t.$$

Recall that $V_t = \bigcup_{a \in \Sigma} V_t(a)$ — the set of data values found in the data tree t.

A *set constraint* is either $\tau = \emptyset$ or $\tau \neq \emptyset$, where τ is a term. A data tree t satisfies $\tau = \emptyset$, written as $t \models \tau = \emptyset$, if and only if $\llbracket \tau \rrbracket_t = \emptyset$ (and likewise for $\tau \neq \emptyset$).

A *linear constraint* ξ over the alphabet Σ is a linear constraint on the variables x_a, for each $a \in \Sigma$ and z_S, for each $S \subseteq \Sigma$. A data tree t satisfies ξ, if ξ holds by interpreting x_a as the number of a-nodes in t, and z_S the cardinality $|[S]_t|$.

THEOREM 6.5. *Given a tree automaton $\mathcal{A}$ and a set $\mathcal{C}$ of set and linear constraints, there exists a weak ODTA $\langle \mathcal{S}, \varphi \rangle$ extended with Presburger constraints such that $\mathcal{L}_{data}(\mathcal{S}, \varphi)$ is precisely the set of ordered-data trees accepted by $\mathcal{A}$ that satisfies all the constraints in $\mathcal{C}$. Moreover, the construction of $\langle S, \varphi \rangle$ takes exponential time in the size of $\mathcal{A}$ and $\mathcal{C}$.*

PROOF. The proof is simply a restatement of the proof in David et al. [2012] into a language of weak ODTA. We need the following notation. For a data term τ, we define a family $\mathbb{S}(\tau)$ of subsets of Σ as follows.

—If $\tau = V(a)$, then $\mathbb{S}(\tau) = \{S \mid a \in S \text{ and } S \subseteq \Sigma\}$.
—If $\tau = \overline{\tau}_1$, then $\mathbb{S}(\tau) = 2^\Sigma - \mathbb{S}(\tau_1)$.
—If $\tau = \tau_1 \star \tau_2$, then $\mathbb{S}(\tau) = \mathbb{S}(\tau_1) \star \mathbb{S}(\tau_2)$, where $\star$ is $\cap$ or $\cup$.

It follows that for every data tree t, we have $\llbracket \tau \rrbracket_t = \bigcup_{S \in \mathbb{S}(\tau)} [S]_t$. Recall that the sets $[S]_t$'s are disjoint.

The desired $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Sigma_0 \rangle$ is defined as follows. The transducer $\mathcal{T}$ is the identity transducer $\mathcal{A}$, and $\Sigma_0 = \emptyset$. The automaton $\mathcal{M}$ accepts a word $v \in (2^\Sigma)^*$ if and only if

C1. for every set constraint $\tau = \emptyset$, v does not contain any symbol from $\mathbb{S}(\tau)$;
C2. for every set constraint $\tau \neq \emptyset$, v contains at least one symbol from $\mathbb{S}(\tau)$.

The formula ξ is the conjunction of all the linear constraints in $\mathcal{C}$.

That $\mathcal{L}_{data}(\mathcal{S}, \xi)$ is indeed precisely the set of ordered-data trees accepted by $\mathcal{A}$ that satisfies all the constraints in $\mathcal{C}$ follows immediately from the definition of $\mathbb{S}$. The exponential upper-bound occurs while constructing the automaton $\mathcal{M}$, which requires the enumeration of each element of 2^Σ and checking both conditions C1 and C2 are satisfied. This completes the proof of Theorem 6.5. □

6.2.3. $FO^2(+1, \prec_{suc})$ over Text. Here we focus our attention on ordered-data words, which can be viewed as trees, where each node has at most one child. We write $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$ to denote ordered-data word in which position i has label a_i and data value d_i. It is called a *text* if all the data values are different and the set of data values $\{d_1, \ldots, d_n\}$ is precisely $\{1, \ldots, n\}$.

It is shown in Manuel [2010] that the satisfiability problem for $\mathrm{FO}^2(+1, \prec_{suc})$ over text is decidable.[7] The following theorem shows that this decidability can be obtained via weak ODTA.

THEOREM 6.6. *For every formula $\varphi \in FO^2(+1, \prec_{suc})$, one can construct effectively a weak ODTA $\mathcal{S}$ such that*

—*For every text w, if $w \in \mathcal{L}_{data}(\varphi)$, then $w \in \mathcal{L}_{data}(\mathcal{S})$.*
—*For every ordered-data word $w \in \mathcal{L}_{data}(\mathcal{S})$, there exists a text $w' \in \mathcal{L}_{data}(\varphi)$ such that $Proj(w) = Proj(w')$.*

The construction of $\mathcal{S}$ takes double exponential time in the size of φ.

PROOF. In Manuel [2010], the decidability is proved by constructing its socalled text automata, also defined in Manuel [2010]. We review the precise definition here. Let $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$ be a text over the alphabet Σ. Therefore, $\mathcal{V}(w) = S_1 \cdots S_n$ is such that each S_i is a singleton.

We define $msp(w)$, the marked string projection of w, as the word $(a_0, b_0) \ldots (a_n, b_n)$, where $b_i \in \{-1, 1, *\}$ and

$$b_i = \begin{cases} -1, & \text{if } 1 \leq i < n \text{ and } d_{i+1} + 1 = d_i, \\ 1, & \text{if } 1 \leq i < n \text{ and } d_i + 1 = d_{i+1}, \\ *, & \text{otherwise.} \end{cases}$$

A text automaton over the alphabet Σ is pair (T_1, T_2), where

—T_1 is a nondeterministic letter-to-letter word transducer with the input alphabet $\Sigma \times \{-1, 1, *\}$ and the output alphabet Γ.
—T_2 is a nondeterministic finite state automaton over Γ.

A text $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$ is accepted by the text automaton (T_1, T_2), if

—$msp(w)$ is accepted by T_1, yielding a string $\alpha_1 \cdots \alpha_n$.
—The string $\alpha_{i_0} \cdots \alpha_{i_n}$ is accepted by T_2, where the indexes $i_1, \ldots, i_n$ are such that $1 = d_{i_1} < d_{i_2} < \cdots < d_{i_n} = n$.

It is shown in Manuel [2010] that for every $\varphi \in \mathrm{FO}^2(+1, \prec_{suc})$, one can construct effectively a text automaton $\mathcal{A}$ such that for every text w, $w \in \mathcal{L}_{data}(\varphi)$ if and only if $w \in \mathcal{L}_{data}(\mathcal{A})$.

Now we are going to show how to get the desired ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma \rangle$. Let (T_1, T_2) be the preceding text automaton. On input ordered-data word $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$, $\mathcal{S}$ performs the following.

—The automaton $\mathcal{T}$ simulates T_1, by guessing $msp(w)$ and outputs its Γ-projection, while store its $\{-1, 1, *\}$-projection in its states.
—The automaton $\mathcal{M}$ is simply T_2.

[7]The definition of text in [Manuel 2010] is slightly different, but it is equivalent to our definition. However, it turns out that the key lemma proved in [Manuel 2010] has a gap which is filled later on in [Figueira 2012b]. The final result is still correct though.

It is straightforward to see that such $\mathcal{S}$ is the desired weak ODTA. The analysis of the complexity is as follows. The construction of the text automaton (T_1, T_2) takes double exponential time in the size of φ. (see [Manuel 2010, Lemmas 5 and 6]). The construction of ODTA $\mathcal{S}$ takes polynomial time in the size of (T_1, T_2). Altogether, it takes double exponential time to construct $\mathcal{S}$ from the original formula φ. This completes the proof of Theorem 6.6. □

7. AN UNDECIDABLE EXTENSION

In this section, we would like to remark on an undecidable extension of ODTA. Recall the language in Example 3.3. It has already noted in the proof of Proposition 5.6 that its complement is not accepted by any ODTA. Formally, the complement of the language in Example 3.3 can be expressed with formula of the form.

$$\forall x\ \forall y\ \bigvee_{a\in\Sigma_0} a(x) \wedge \bigvee_{a\in\Sigma_0} a(y) \wedge {E_\downarrow}^*(x,y) \rightarrow x \prec y, \tag{8}$$

where $\Sigma_0 \subseteq \Sigma$ and ${E_\downarrow}^*$ denotes the transitive closure of $E_\downarrow$. In the following, we are going to show that given an ODTA and a collection $\mathcal{C}$ of formulas of the form of Eq. (8), it is undecidable to check whether there is an ordered-data tree $t \in \mathcal{L}_{data}(\mathcal{S})$ such that $t \models \psi$, for all $\psi \in \mathcal{C}$.

The proof is simply an observation that the proof of Bojanczyk et al. [2011a, Proposition 29] can be applied directly here which it is proved that the satisfiability of $\mathrm{FO}^2(E_\downarrow, {E_\downarrow}^*, \sim, \prec)$ is undecidable.[8] The reduction is from the Post Correspondence Problem (PCP), where given an instance of PCP, one can effectively construct a formula of the form $\varphi \wedge \psi$, where $\varphi \in \mathrm{FO}^2(E_\downarrow, {E_\downarrow}^*, \sim)$ and ψ is a formula of the form of Eq. (8). Since φ can be captured by ODTA, the undecidability of ODTA extended with formulas of the form of Eq. (8) follows immediately.

At this point, we would also like to point out that extending ODTA with operation such as addition on data values will immediately yield undecidability. This can be deduced immediately from Halpern [1991], where we know that together with unary predicates, addition yields undecidability.

8. WHEN THE DATA VALUES ARE STRINGS

In this section, we discuss data trees where the data values are strings from $\{0,1\}^*$, instead of natural numbers. We call such trees *string data trees*. There are two common kinds of order for strings: the prefix order and the lexicographic order. Strings with lexicographic order are simply linearly ordered domain, thus, ODTA can be applied directly in such case.

For the prefix order, we have to modify the definition of ODTA. Consider a string data tree t over the alphabet Σ. Let V_t be the set of data values found in t. We define $\mathcal{V}_\Sigma(t)$ as a *tree* over the alphabet 2^Σ, where

—$\mathsf{Dom}(\mathcal{V}_\Sigma(t))$ is $V_t \cup \{\epsilon\}$;
—for $u, v \in \mathsf{Dom}(\mathcal{V}_\Sigma(t))$, u is a parent of v if u is a prefix of v and there is no $w \in \mathsf{Dom}(\mathcal{V}_\Sigma(t))$ such that u is a prefix of w and w is a prefix of v;
—for $u \in \mathsf{Dom}(\mathcal{V}_\Sigma(t))$ the label of u is S, if $u \in [S]_t$; and ROOT, if $u = \epsilon$.

[8] Technically, the undecidability in Bojanczyk et al. [2011a, Proposition 29] is proved on data strings over the logic $\mathrm{FO}^2(+1, <, \sim, \prec)$, which of course, is equivalent to $\mathrm{FO}^2(E_\downarrow, {E_\downarrow}^*, \sim, \prec)$.

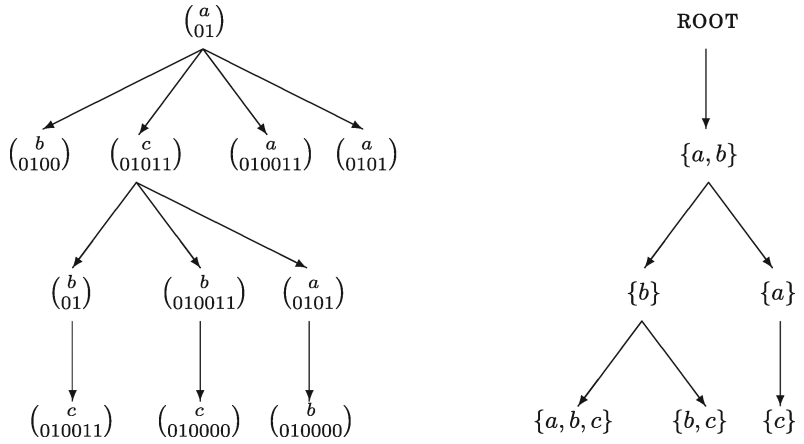

Fig. 2. An example of a string data tree (on the left) and the tree representation of its data values (on the right).

We call $\mathcal{V}_\Sigma(t)$ the *tree representation* of the data values in t. Consider an example of a string data tree in Figure 2. We have

$$[\{a\}]_t = \{0101\} \qquad [\{b\}]_t = \{0100\}$$
$$[\{c\}]_t = \{01011\} \qquad [\{a, b\}]_t = \{01\}$$
$$[\{b, c\}]_t = \{01000\} \quad [\{a, b, c\}]_t = \{010011\}.$$

So $\mathsf{Dom}(\mathcal{V}_\Sigma(t)) = \{01, 0100, 0101, 010011, 010000, 01011\}$, and

—01 is the parent of 0100 and 0101.
—0100 is the parent of 010011 and 010000.
—0101 is the parent of 01011.

Now an ODTA for string data trees is $\mathcal{S} = \langle \mathcal{T}, \mathcal{A}, \Gamma_0 \rangle$, where $\mathcal{T}$ is a letter-to-letter transducer from $\Sigma \times \{\top, \bot, *\}^3$ to Γ; $\mathcal{A}$ is an unranked tree automaton over the alphabet 2^Γ; $\Gamma_0 \subseteq \Gamma$. The requirement for acceptance is the same as in Section 5, except that $\mathcal{A}$ takes a tree over the alphabet 2^Γ as the input.

We observe that in the proof of the decidability of the non-emptiness of ODTA $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$, the automaton $\mathcal{M}$ is converted in polynomial time into a Presburger formula by applying Proposition 2.3, which actually holds for tree automata. Hence, the decision procedures in Sections 5 and 6 can also be applied to string data trees.

9. CONCLUDING REMARKS

In this article, we study data trees in which the data values come from a linearly ordered domain, where in addition to equality test, we can test whether the data value in one node is greater than the other. We introduce ordered-data tree automata (ODTA), provide its logical characterisation, and prove that its non-emptiness problem is decidable. We also show the logic $\exists\mathrm{MSO}^2(E_\downarrow, E_\rightarrow, \sim)$ can be captured by ODTA.

Then we define weak ODTA, which essentially are ODTA without the ability to perform equality test on data values on two adjacent nodes. We provide its logical characterisation. We show that a number of existing formalisms and models studied in the literature so far can be captured already by weak ODTA. We also show that the definition of ODTA can be easily modified, to the case where the data values come from a partially ordered domain, such as strings.

We believe that the notion of ODTA provides new techniques to reason about ordered-data values on unranked trees, and thus, can find potential applications in practice. We also prove that ODTA capture various formalisms on data trees studied so far in the literature. As far as we know this is the first formalism for data trees with neat logical and automata characterisations.

ACKNOWLEDGMENTS

The author thanks Egor V. Kostylev for his careful proofreading, as well as Nadime Francis for pointing out the reference Halpern [1991]. Finally, the author thanks the anonymous referees for both the conference and the journal versions for their careful reading and comments which greatly improve the article.

REFERENCES

N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. 2003. XML with data values: Typechecking revisited. *J. Comput. Syst. Sci.* 66, 4, 688–727.

M. Arenas, W. Fan, and L. Libkin. 2008. On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.* 38, 3, 841–880.

M. Benedikt, C. Ley, and G. Puppis. 2010. Automata vs. logics on data words. In *Proceedings of the 24th International Workshop on Computer Science Logic (CSL'10)*. Lecture Notes in Computer Science, vol. 6247, Springer-Verlag, Berlin, 110–124.

H. Björklund and M. Bojanczyk. 2007. Bounded depth data trees. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*. Lecture Notes in Computer Science, vol. 4596, Springer-Verlag, Berlin, 862–874.

H. Björklund, W. Martens, and T. Schwentick. 2008. Optimizing conjunctive queries over trees using schema information. In *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS'08)*. Lecture Notes in Computer Science, vol. 5162, Springer-Verlag, Berlin, 132–143.

M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. 2011a. Two-variable logic on data words. *ACM Trans. Comput. Logic* 12, 4, 27.

M. Bojanczyk, B. Klin, and S. Lasota. 2011b. Automata with group actions. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS'11)*. 355–364.

M. Bojanczyk, A. Muscholl, T. Schwentick, and L. Segoufin. 2009. Two-variable logic on data trees and XML reasoning. *J. ACM 56,* 3.

P. Bouyer, A. Petit, and D. Thérien. 2001. An algebraic characterization of data and timed languages. In *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'01)*. Lecture Notes in Computer Science, vol. 2154, Springer-Verlag, Berlin, 248–261.

H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 2007. Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata. (Last accessed 10/07).

C. David, L. Libkin, and T. Tan. 2010. On the satisfiability of two-variable logic over data words. In *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-17).* Lecture Notes in Computer Science, vol. 6397, Springer-Verlag, Berlin, 248–262.

C. David, L. Libkin, and T. Tan. 2012. Efficient reasoning about data trees via integer linear programming. *ACM Trans. Datab. Syst.* 37, 3, 19.

S. Demri, D. D'Souza, and R. Gascon. 2007. A decidable temporal logic of repeating values. In *Proceedings of the International Symposium on Logical Foundations of Computer Science (LFCS'07).* Lecture Notes in Computer Science, vol. 4514, Springer-Verlag, Berlin, 180–194.

S. Demri and R. Lazić. 2009. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Logic* 10, 3.

A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. 2009. Automatic verification of data-centric business processes. In *Proceedings of the 12th International Conference on Database Theory (ICDT'09)*. 252–267.

W. Fan and L. Libkin. 2002. On XML integrity constraints in the presence of DTDs. *J. ACM* 49, 3, 368–406.

D. Figueira. 2009. Satisfiability of downward XPath with data equality tests. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'09)*. 197–206.

D. Figueira. 2011. A decidable two-way logic on data words. In *Proceedings of the 26th Annual Symposium on Logic in Computer Science (LICS'11)*. 365–374.

D. Figueira. 2012a. Alternating register automata on finite data words and trees. *Logic. Methods Comput. Sci.* 8, 1.

D. Figueira. 2012b. Satisfiability for two-variable logic with two successor relations on finite linear orders. http://arxiv.org/abs/1204.2495.

D. Figueira, P. Hofman, and S. Lasota. 2010. Relating timed and register automata. In *Proceedings of the 17th International Workshop on Expressiveness in Concurrency (EXPRESS'10)*.

D. Figueira and L. Segoufin. 2011. Bottom-up automata on data trees and vertical XPath. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS'11)*.

O. Grumberg, O. Kupferman, and S. Sheinvald. 2010. Variable automata over infinite alphabets. In *Proceedings of the 4th International Conference on Language and Automata Theory and Applications (LATA'10)*. Lecture Notes in Computer Science, vol. 6031, Springer-Verlag, Berlin, 561–572.

J. Halpern. 1991. Presburger arithmetic with unary predicates is π_1 complete. *J. Symbol. Logic* 56, 2, 637–642.

M. Jurdzinski and R. Lazic. 2011. Alternating automata on data trees and XPath satisfiability. *ACM Trans. Comput. Logic* 12, 3, 19.

M. Kaminski and N. Francez. 1994. Finite-memory automata. *Theor. Comput. Sci.* 134, 2, 329–363.

A. Kara, T. Schwentick, and T. Tan. 2012. Feasible automata for two-variable logic with successor on data words. In *Proceedings of the 6th International Conference on Language and Automata Theory and Applications (LATA'12)*. Lecture Notes in Computer Science, vol. 7183, Springer-Verlag, Berlin, 351–362.

R. Lazić. 2011. Safety alternating automata on data words. *ACM Trans. Comput. Logic* 12, 2, 10.

L. Libkin. 2004. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science, Springer.

A. D. Manuel. 2010. Two variables and two successors. In *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10)*. Lecture Notes in Computer Science, vol. 6281, Springer-Verlag, Berlin, 513–524.

F. Neven. 2002. Automata, logic, and XML. In *Proceedings of the 16th International Workshop on Computer Science Logic (CSL'02)*. Lecture Notes in Computer Science, vol. 2471, Springer-Verlag, Berlin, 2–26.

F. Neven, T. Schwentick, and V. Vianu. 2004. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic* 5, 3, 403–435.

T. Schwentick. 2004. XPath query containment. *SIGMOD Rec.* 33, 1, 101–109.

T. Schwentick and T. Zeume. 2010. Two-variable logic with two order relations. In *Proceedings of the 24th International Workshop on Computer Science Logic (CSL'10)*. Lecture Notes in Computer Science, vol. 6247, Springer-Verlag, Berlin, 499–513.

L. Segoufin and S. Torunczyk. 2011. Automata based verification over linearly ordered data domains. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS'11)*.

H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. 2004. Counting in trees for free. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*. Lecture Notes in Computer Science, vol. 3142, Springer-Verlag, Berlin, 1136–1149.

J. Thatcher. 1967. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *J. Comput. Syst. Sci.* 1, 4, 317–322.

W. Thomas. 1997. Languages, automata, and logic. In *Handbook of Formal Languages,* vol. 3*, Beyond Words*, Springer, 389–455.

K. N. Verma, H. Seidl, and T. Schwentick. 2005. On the complexity of equational Horn clauses. In *Proceedings of the 20th International Conference on Automated Deduction (CADE-20)*. Lecture Notes in Computer Science, vol. 3632, Springer-Verlag, Berlin, 337–352.

Received February 2013; revised July, September 2013; accepted September 2013