2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science

An Automata Model for Trees with Ordered Data Values

Tony Tan
University of Edinburgh
Edinburgh, UK
Email: ttan@inf.ed.ac.uk

University of Warsaw
Warsaw, Poland

***Abstract*—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 paper 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 emptiness problem is decidable in 3-NEXPTIME. We also show that the two-variable logic on unranked trees, studied by Bojanczyk, Muscholl, Schwentick and Segoufin in 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 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.

***Index Terms*—Automata, data unranked trees, logic, two-variable logic, ordered data values.**

I. 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 [1], [2], [5], [12], [17], [18], [32], as well as system specifications [9], [13], [36], where many properties simply cannot be captured by finite alphabets. This has motivated various works on data words [3], [8], [14], [22], [25], [33], as well as on data trees [4], [6], [20], [21], [24]. 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., [8], [17], [33].

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., [2], [5], [34], [17], [18]). The second looks for sufficiently general automata models that can express reasoning tasks of interest, but are still decidable (see, e.g., [14], [6], [24], [36]).

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 [15] 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 [31], [35], [36] are on words, while in most applications we need to consider trees. Moreover, these works are incomparable to some interesting existing formalisms [17], [6], [2], [12], [24], [14], [29] known to be able to capture various interesting scenarios common in practice. On top of that many useful techniques, notably those introduced in [17], [8], [6], [24], 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, most known decidability results are lost as soon as we add the order relation on data values. See, e.g., [8].

In this paper 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 paper is the first step, where we introduce an automata model for ordered-data trees, provide its logical characterisation, and prove that it has decidable emptiness problem. Moreover, we also show that it can capture various well known formalisms.

Brief description of the results in this paper: 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

1043-6871/12 $26.00 © 2012 IEEE
DOI 10.1109/LICS.2012.69

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; the formula 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; and the formula 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 [6], 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 (1) in which ψ is a true formula. We then prove that the 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 III.) 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 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 (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, i.e. those in [17], [12], [31].

We should remark that [12] studies a formalism which consists of tree automata and a collection of *set* and *linear* constraints.* It is shown that the satisfaction problem of such formalism is NP-complete. In fact, it is also shown in [12] that a single set constraint (without tree automaton and linear constraint) already yields NP-hardness. Weak ODTA are essentially equivalent to the formalism in [12] extended with the full expressive power of the first-order logic $\text{FO}(\sim, \prec, \prec_{suc})$. It is worth to note that despite such extension, the 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 [11] and [27]. However, it must be noted that [11], [27] 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. In the paper [8] the model *data automata* was introduced, 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 using two variables only 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 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 [11] 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 satisfaction problem of $\exists\text{MSO}^2(+1, \sim)$ on strings. Recently in [27] 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 [14]. Intuitively, these are LTL formulas equipped with a finite number of registers to store the data values. We denote by $\text{LTL}_n^{\downarrow}[\mathtt{X}, \mathtt{U}]$, 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 $\text{LTL}_n^{\downarrow}[\mathtt{X}, \mathtt{U}]$ languages and the emptiness problem for alternating RA_1 is decidable. However, the complexity is non primitive recursive. Hence, the satisfiability problem for $\text{LTL}_1^{\downarrow}(\mathtt{X}, \mathtt{U})$ is decidable as well. Adding one more register or past time operators, such as $\mathtt{X}^{-1}$ or $\mathtt{U}^{-1}$, to $\text{LTL}_1^{\downarrow}(\mathtt{X}, \mathtt{U})$ makes the satisfiability problem undecidable. In [29] a weaker version of alternating RA_1, called safety alternating RA_1, is considered, and the emptiness problem is shown to be EXPSPACE-complete.

A model for data words with linearly ordered data values was proposed in [36]. 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 emptiness problem for this model is decidable in PSPACE. However, no logical characterisation is provided for such model.

In [7] another type of register automata for words was in-

*We will later define formally what set and linear constraints are.

troduced and studied, which is a generalisation of the original register automata introduced by Kaminski and Francez [25], 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. This model is based on the notion of monoid for data words, and is incomparable with our model here.

It is shown in the paper [31] that the satisfaction problem for $\text{FO}^2(+1, \prec_{suc})$ 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 satisfaction problem for $\text{FO}^2(+1, \prec_{suc})$ can be reduced to the emptiness problem of our model.

In [35] it is shown that the satisfaction problem of the logic $\text{FO}^2(<, \prec)$ on *words* is decidable. This logic is incomparable with our model. However, it should be noted that $\text{FO}^2(<)$ *cannot* capture the whole class of regular languages.

The work on data trees that we are aware of is in [6], [24]. In [6] it was shown that the satisfaction problem for the logic $\exists\text{MSO}^2(E_{\downarrow}, E_{\rightarrow}, \sim)$ 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 [24] 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 emptiness problem, though decidable, is non primitive recursive.

Organisation: This paper is organised as follows. In Section II we give some preliminary background. In Section III we formally define the logic for ordered-data trees and present a few examples as well as notations that we need in this paper. In Section IV 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 V and weak ODTA in Section VI. In Section VII we discuss a couple of the undecidable extensions of weak ODTA. In Section VIII 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 IX.

II. 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.

A. 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 $b_{\sigma(1)} \cdots b_{\sigma(m)} \in \mathcal{L}$.

Theorem 1: [16, Corollary 2.2] *A language is commutative and regular if and only if it is a finite union of periodic languages.*

B. 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, i.e. 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* [10], [38] 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 $\rho_{\mathcal{A}}(\epsilon) \in F$, i.e., 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.

C. 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: [37], [40] *The 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 3: [37], [40] *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*

- *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 $\mathsf{Parikh}(t) = \bar{n}$.*

III. Ordered-dataTTrees 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 $\mathbb{N} = \{0, 1, \ldots\}$.[†]

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 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.

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

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 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.

A. String representations of data values

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

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

Note that for each $a \in \Gamma$,

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

Since the sets $[S]_t$'s are disjoint, it is immediate that $|V_t(a)| = \sum_{S \text{ s.t. } a \in S} |[S]_t|$.

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 [11], [12], 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 $S_4 = \{a, c\}$.

B. 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

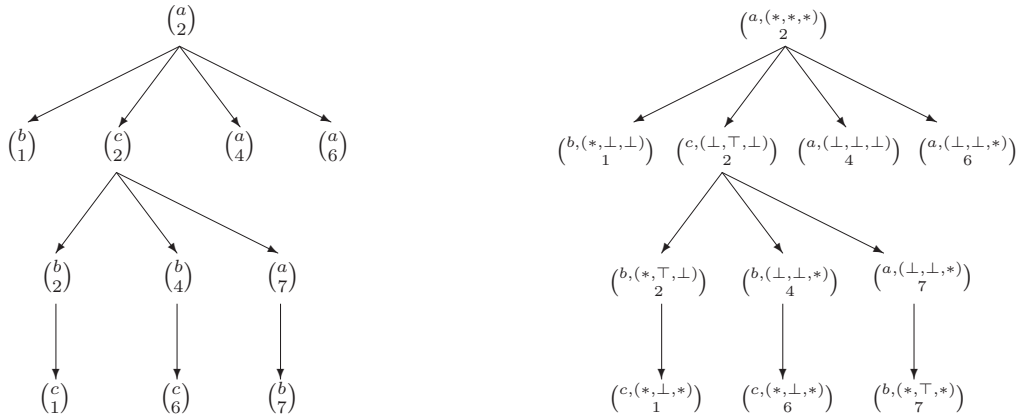

Fig. 1. An example of an ordered-data tree (on the left) and its profile (on the right).

- the relations $\{a(\cdot)\}_{a\in\Sigma}, E_\downarrow, E_\rightarrow$ are as defined before in Subsection II-B,
- $u \sim v$ holds, if $val_t(u) = val_t(v)$,
- $u \prec v$ holds, if $val_t(u) < val_t(v)$,
- $u \prec_{suc} v$ holds, if $val_t(v)$ is the minimal data value in t greater than $val_t(u)$.

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, i.e., 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, i.e., 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 [39].

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 $\mathcal{L}_{data}(\varphi)$.[‡]

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.

[‡]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.

Theorem 4: (See, for example, [8], [17], [33]) *The satisifaction problem for 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 5: [6] *The satisfaction problem for $\exists$MSO$^2(E_\downarrow, E_\rightarrow, \sim)$ is decidable.*

C. A few examples

In this subsection we present a few examples of properties of ordered-data trees expressible in our logic. Some of the examples are special cases of more general techniques that will be used later on.

Example 6: 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 $v \prec u$.

Example 7: For a fixed set $S \subseteq \Sigma$ and an integer $m \geq 1$, we consider the language $\mathcal{L}^{S,m}_{data}$ such that $t \in \mathcal{L}^{S,m}_{data}$ if and only if $|[S]_t| = m$.

We pick an arbitrary symbol $a \in S$. The language $\mathcal{L}^{S,m}_{data}$ can be expressed in $\exists$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
  - for each $b \in S$, there is a b-node with the same data value; and
  - the data value cannot be found in any b-node, for every $b \notin S$.

Example 8: 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 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 9: 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:

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

IV. 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.

A. 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 $V_G(a)$ the set of data values found in a-nodes in G.

Lemma 10: 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 $a \in \Gamma$, $V_G(a) = V_{G'}(a)$;*
(3) *for each $u, v \in V(G)$, if $(u, v) \in E(G')$, then $\mathrm{va}\ell_{G'}(u) \neq \mathrm{va}\ell_{G'}(v)$.*

Note that in Lemma 10, 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.

B. 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, if $f(a) \in \{1, 2, \ldots, k\}$, for all $a \in S$, and $f(a) = 0$, for all $a \notin S$.

Let t be an ordered-data tree and $d_1 < \cdots < d_m$ be the data values found in t. The *k-extended representation* of t is the string $\mathcal{V}_{\Gamma}^k(t) = (S_1, f_1) \cdots (S_m, f_m) \in 2^{\Gamma} \times \mathcal{F}_{\Gamma,k}$ such that $S_1 \cdots S_m = \mathcal{V}_{\Gamma}(t)$ 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-nodes in t with data value d_i,
3) if $f_i(a) = k$, then there are at least k number of a-nodes in t with data value d_i.

We assume that in every formula in $\mathrm{MSO}(\sim, \prec, \prec_{suc})$ all the monadic second-order quantifiers precede the first-order part. That is, sentences in $\mathrm{MSO}(\sim, \prec, \prec_{suc})$ are of the form: $\varphi := Q_1 X_1 \cdots Q_s X_s\ \psi$, where the X_i's are monadic second-order variables, the Q_i's are $\exists$ or $\forall$ and $\psi \in \mathrm{FO}(\sim, \prec, \prec_{suc})$ 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 11: Let t_1 and t_2 be ordered-data trees over Γ such that $\mathcal{V}_\Gamma^{k2^s}(t_1) = \mathcal{V}_\Gamma^{k2^s}(t_2)$. For any MSO$(\sim, \prec, \prec_{suc})$ sentence φ such that MSO-qr$(\varphi) \leq s$ and FO-qr$(\varphi) \leq k$, $t_1 \models \varphi$ if and only if $t_2 \models \varphi$.

V. 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 12: 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 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 6, 7, 8 and 9.

Example 13: An ODTA $\mathcal{S}^a = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ that accepts the language $\mathcal{L}^a_{data}$ in Example 6 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 that 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 14: 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 7 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 15: An ODTA $\mathcal{S}^{S,\ (\mathrm{mod}\ m)} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ that accepts the language $\mathcal{L}^{S,\ (\mathrm{mod}\ m)}_{data}$ in Example 8 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 16: An ODTA $\mathcal{S}^{a*} = \langle \mathcal{T}, \mathcal{M}, \Gamma_0 \rangle$ that accepts the language $\mathcal{L}^{a*}_{data}$ in Example 9 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. Oftentimes models that are closed under negation have undecidable emptiness/satisfaction problem.

Proposition 17: 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 $\mathcal{S}_2$.

That ODTA languages are not closed under negation follows from the fact that the negation of the language in Example 13 is not accepted by ODTA. The proof is rather straightforward, thus, omitted. ■

We should remark that in Section VII we will discuss that extending ODTA with the complement of languages of the form in Example 13 will immediately yield undecidability.

Next we give the ODTA characterisation of the logic $\exists\mathrm{MSO}^2(E_\downarrow, E_\rightarrow, \sim)$.

Theorem 18: 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.

The next result is the logical characterisation of ODTA.

Theorem 19: 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 $X_1, \ldots, X_m$.

Finally, we show that the emptiness problem for ODTA is decidable. The decision procedure in Theorem 20 runs in 3-NExpTime, while the decision procedure for $\exists\mathrm{MSO}^2(E_\downarrow, E_\rightarrow, \sim)$ proposed in [6] also runs in 3-NExpTime. However, we should remark that if we use our algorithm for the satisfaction problem of $\exists\mathrm{MSO}^2(E_\downarrow, E_\rightarrow, \sim)$ via the translation in Theorem 18, the complexity will jump to 5-NExpTime, since there is a double exponential blow-up in the translation.

Theorem 20: The emptiness problem for ODTA is decidable.

VI. 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 $\mathcal{V}_\Gamma(t')$; and
- 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 21: The emptiness problem for weak ODTA is in NP.

Next, we give the logical characterisation of weak ODTA.

Theorem 22: 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 22 is the same as the proof of Theorem 19. The difference is that to simulate the FO$^2(E_\downarrow, E_\rightarrow)$ formula φ, the profile information is not necessary.

A. 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 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 $\xi(x_1, \ldots, x_\ell, y_1, \ldots, y_{2^\ell})$ is an existential Presburger formula with the free variables $x_1, \ldots, x_\ell, y_1, \ldots, y_{2^\ell - 1}$. A 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 $\langle \mathcal{S}, \xi \rangle$.

It should be immediate that the emptiness problem of weak ODTA extended with Presburger constraint is still decidable in NP.

B. Comparison with other known decidable formalisms

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

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 context free 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 constraints* are constraints of the form:
$$\forall x \forall y (a(x) \wedge a(y) \wedge x \sim y \rightarrow x = y),$$
denoted by $key(a)$.
- The *inclusion constraints* are constraints of the form:
$$\forall x \exists y (a(x) \rightarrow b(y) \wedge x \sim y),$$
denoted by $V(a) \subseteq V(b)$.

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 $\mathcal{C}$. In [17] it is shown that this problem is NP-complete.

Theorem 23: 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 $\mathcal{C}$.

It must be noted that our construction in Theorem 23 outputs an automaton $\mathcal{M}$ 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 24: Given a DTD D and a collection $\mathcal{C}$ of integrity constraints, one can construct a weak ODTA $\mathcal{S}$ in non-deterministic 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 $\mathcal{C}$.

2) Set and linear constraints for data trees: In the paper [12] 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 word t:

$$[\![V(a)]\!]_t = V_t(a) \qquad [\![\overline{\tau}]\!]_t = V_t - [\![\tau]\!]_t$$
$$[\![\tau_1 \cap \tau_2]\!]_t = [\![\tau_1]\!]_t \cap [\![\tau_2]\!]_t \qquad [\![\tau_1 \cup \tau_2]\!]_t = [\![\tau_1]\!]_t \cup [\![\tau_2]\!]_t$$

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 $[\![\tau]\!]_t = \emptyset$ (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 25: 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 $\mathcal{C}$.

Its proof is simply a restatement of the proof in [12] into a language of weak ODTA.

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 the paper [31] that the satisfaction problem for FO$^2(+1, \prec_{suc})$ over text is decidable.[§] The following theorem shows that this decidability can be obtained via weak ODTA.

Theorem 26: 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 $\mathsf{Proj}(w) = \mathsf{Proj}(w')$.*

VII. An Undecidable Extension

In this section we would like to remark on an undecidable extension of weak ODTA. Recall the language in Example 6. It has already noted in the proof of Proposition 17 that its complement is not accepted by any ODTA. Formally, the complement of the language in Example 6 can be expressed with formula of the form:

$$\forall x \, \forall y \bigvee_{a \in \Sigma_0} a(x) \wedge \bigvee_{a \in \Sigma_0} a(y) \wedge {E_\downarrow}^*(x, y) \rightarrow x \prec y, \quad (2)$$

where $\Sigma_0 \subseteq \Sigma$ and ${E_\downarrow}^*$ denotes the transitive closure of $E_\downarrow$. It can already be deduced from [8, Proposition 29] that given an ODTA and a collection $\mathcal{C}$ of formulas of the form (2), 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 $\psi \in \mathcal{C}$.

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 [23] where we know that together with unary predicates, addition yields undecidability.

VIII. 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 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 $u = \epsilon$.

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

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

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; and
- 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 V, except that $\mathcal{A}$ takes a tree over the alphabet 2^Γ as the input. All the results in Sections V and VI can be carried over immediately to this model.

IX. Concluding Remarks

In this paper 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 emptiness problem is decidable. We also show the logic $\exists$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.

[§]The definition of text in [31] is slightly different, but it is equivalent to our definition. However, it turns out that the key lemma proved in [31] has a serious gap, which is filled later on in [19]. The final result is still correct though.

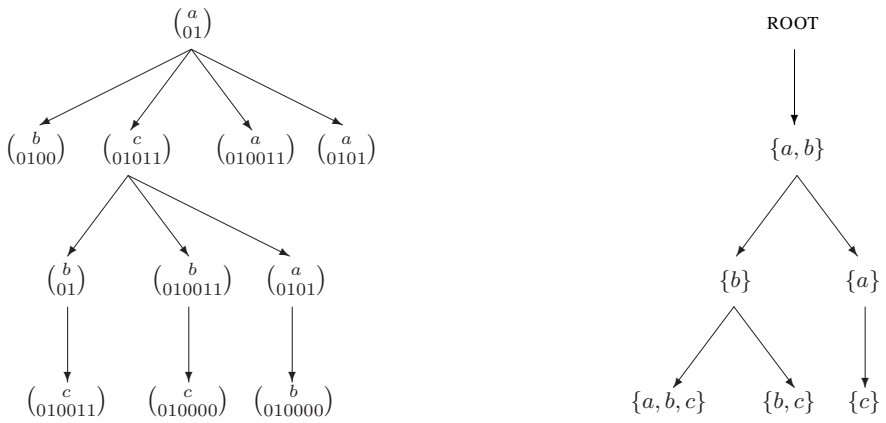

Fig. 2. An example of a string data tree (on the left) and the tree representation of its data values (on the right).

Acknowledgment. Work is supported by FET-Open Project FoX, grant agreement 233599. The author would like to thank Egor V. Kostylev for careful proof reading of this paper and for many useful suggestions to improve it. The author also thanks Leonid Libkin and Claire David for helpful discussions, and Nadime Francis for pointing out the reference [23]. The author was also supported by the *Querying and Managing Navigational Databases* project realized within the Homing Plus programme of the Foundation for Polish Science, co-financed by the European Union from the Regional Development Fund within the Operational Programme Innovative Economy ("Grants for Innovation"). Finally, the author also thanks the anonymous referees for their careful reading and comments.

REFERENCES

[1] N. Alon, T. Milo, F. Neven, D. Suciu, V. Vianu. XML with data values: typechecking revisited. *J. Comput. Syst. Sci.* 66(4): 688-727 (2003).
[2] M. Arenas, W. Fan, L. Libkin. On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.* 38(3): 841-880 (2008).
[3] M. Benedikt, C. Ley, G. Puppis. Automata vs. Logics on Data Words. In *CSL 2010*.
[4] H. Björklund, M. Bojanczyk. Bounded depth data trees. In *ICALP 2007*.
[5] H. Björklund, W. Martens, T. Schwentick. Optimizing conjunctive queries over trees using schema information. In *MFCS 2008*.
[6] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM* 56(3), 2009.
[7] M. Bojanczyk, B. Klin, S. Lasota. Automata with Group Actions. In *LICS* 2011.
[8] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on data words. *ACM TOCL* 2011.
[9] P. Bouyer, A. Petit, D. Thérien. An algebraic characterization of data and timed languages. In *CONCUR 2001*.
[10] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, M. Tommasi. *Tree Automata: Techniques and Applications*. 2007.
[11] C. David, L. Libkin, T. Tan. On the Satisfiability of Two-Variable Logic over Data Words. In *LPAR 2010*.
[12] C. David, L. Libkin, T. Tan. Efficient Reasoning about Data Trees via Integer Linear Programming. In *ICDT 2011*.
[13] S. Demri, D. D'Souza, R. Gascon. A Decidable Temporal Logic of Repeating Values. In *LFCS 2007*.
[14] S. Demri, R. Lazic. LTL with the freeze quantifier and register automata. *ACM TOCL* 10(3), 2009.
[15] A. Deutsch, R. Hull, F. Patrizi, V. Vianu. Automatic verification of data-centric business processes. In *ICDT* 2009.
[16] A. Ehrenfeucht, G. Rozenberg. Commutative Linear Languages. Technical Report CU-CS-209-81, June 1981.
[17] W. Fan, L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM* 49(3): 368–406 (2002).
[18] D. Figueira. Satisfiability of downward XPath with data equality tests. In *PODS 2009*.
[19] D. Figueira. Forward-XPath and extended register automata on data-trees. In `http://arxiv.org/abs/1204.2495`.
[20] D. Figueira. Satisfiability for two-variable logic with two successor relations on finite linear orders. In *ICDT 2010*.
[21] D. Figueira, L. Segoufin. Bottom-up automata on data trees and vertical XPath. In *STACS 2011*.
[22] O. Grumberg, O. Kupferman, S. Sheinvald. Variable Automata over Infinite Alphabets. In *LATA 2010*.
[23] J. Y. Halpern. Presburger Arithmetic with Unary Predicates is Π^1_1 Complete. *Journal of Symbolic Logic*, 56(2), 1991.
[24] M. Jurdzinski, R. Lazic. Alternation-free modal mu-calculus for data trees. In *LICS 2007*.
[25] M. Kaminski, N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2): 329–363 (1994).
[26] M. Kaminski, T. Tan. Tree automata over infinite alphabets. In *Pillars of Computer Science*, 2008.
[27] A. Kara, T. Schwentick, T. Tan. Feasible Automata for Two-Variable Logic with Successor on Data Words. In *LATA* 2012.
[28] A. Kara, T. Schwentick, T. Zeume. Temporal Logics on Words with Multiple Data Values. In *FSTTCS* 2010.
[29] R. Lazic. Safety alternating automata on data words. *ACM TOCL* (12)2, 2011.
[30] L. Libkin. *Elements of Finite Model Theory*. Springer 2004.
[31] A. Manuel. Two orders and two variables. In *MFCS 2010*.
[32] F. Neven. Automata, logic, and XML. In *CSL 2002*.
[33] F. Neven, Th. Schwentick, V. Vianu. Finite state machines for strings over infinite alphabets. *ACM TOCL* 5(3), 403–435 (2004).
[34] T. Schwentick. XPath query containment. *SIGMOD Record* 33(1): 101-109 (2004).
[35] T. Schwentick, T. Zeume. Two-Variable Logic with Two Order Relations. In *CSL 2010*.
[36] L. Segoufin, S. Torunczyk. Automata based verification over linearly ordered data domains. In *STACS 2011*.
[37] H. Seidl, Th. Schwentick, A. Muscholl, P. Habermehl. Counting in trees for free. In *ICALP 2004*.
[38] J. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *JCSS 1*, 1967.
[39] W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages, Vol. 3*, Springer, 1997, pages 389–455.
[40] K. Verma, H. Seidl, T. Schwentick. On the complexity of equational horn clauses. In *CADE 2005*, pages 337–352.