Bottom-up automata on data trees and vertical XPath*

Diego Figueira and Luc Segoufin

INRIA and ENS Cachan, LSV

Abstract

A data tree is a tree whose every node carries a label from a finite alphabet and a datum from some infinite domain. We introduce a new model of automata over unranked data trees with a decidable emptiness problem. It is essentially a bottom-up alternating automaton with one register, enriched with epsilon-transitions that perform tests on the data values of the subtree. We show that it captures the expressive power of the vertical fragment of XPath —containing the child, descendant, parent and ancestor axes— obtaining thus a decision procedure for its satisfiability problem.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Decidability, XPath, Data trees, Bottom-up tree automata

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

We study formalisms for data trees. A data tree is a tree where each position carries a label from a finite alphabet and a *datum* from some infinite domain. This structure has been considered in the realm of semistructured data, timed automata, program verification, and generally in systems manipulating data values. Finding decidable logics or automaton models over data trees is an important quest when studying data-driven systems.

A data tree can model an XML document. One wants to decide, for example, if two properties of XML documents expressed in some formalism are equivalent. This problem is usually equivalent to the satisfiability problem. One such formalism to express properties of XML documents is the logic XPath. Although satisfiability of XPath in the presence of data values is undecidable, there are some known decidable data-aware fragments [4, 5, 1, 3]. Here, we investigate a rather big fragment that nonetheless is decidable. *Vertical*-XPath is the fragment that contains all downward and upward axes, but no *horizontal* axis is allowed.

We introduce a novel automaton model that captures vertical-XPath. We show that the automaton has a decidable emptiness problem and therefore that the satisfiability problem of vertical-XPath is decidable. The **B**ottom-**U**p **D**ata **A**utomata (or BUDA) are bottom-up alternating tree automata with one register to store and compare data values. Further, these automata can compare the data value currently stored in the register with the data value of a descendant node, reached by a downward path satisfying a given regular property. Hence, in some sense, it has a two-way behavior. However, they cannot test horizontal properties on the siblings of the tree, like "the root has exactly three children".

Our main technical result shows the decidability of the emptiness problem of this automaton model. We show this through a reduction to the the coverability problem of a well-structured transition system (WSTS [8]). Each BUDA automaton is associated with a

* This research was funded by the ERC research project FoX under grant agreement FP7-ICT-233599.

Conference title on which this volume is based on.
Editors: Editor; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

transition system, in such a way that a derivation in this transition system corresponds to a run of the automaton, and vice-versa. The domain of the transition system consists in the *abstract configurations* of the automaton, which contains all the information necessary to preserve from a (partial) bottom-up run of the automaton in a subtree in order to continue the simulation of the run from there. On the one hand we show that BUDA can be simulated using an appropriate transition relation on sets of abstract configurations. On the other hand, we exhibit a well-quasi-order (wqo) on those abstract configurations and show that the transition relation is "monotone" relative to this wqo. This makes the coverability problem (and hence the emptiness problem) decidable [8].

In terms of expressive power, we show that BUDA can express any node expression of the vertical fragment of XPath. Core-XPath (term coined in [10]) is the fragment of XPath 1.0 that captures its navigational behavior, but cannot express any property involving data. It is easily shown to be decidable. The extension of this language with the possibility to make equality and inequality tests between data values is named Core-Data-XPath in [3], and it has an undecidable satisfiability problem [9]. By "vertical XPath" we denote the fragment of Core-Data-XPath that can only use the downward axes CHILD and DESCENDANT and the upward axes PARENT and ANCESTOR (no navigation among siblings is allowed). It follows that vertical XPath is decidable, settling an open question [2, Question 5.10].

Related work. A model of *top-down* tree automata with one register and alternating control (ATRA) is introduced in [12], where the decidability of its emptiness problem is proved. ATRA are used to show the decidability of temporal logics extended with a "freeze" operator. This model of automata was extended in [5] with the name ATRA(guess, spread) in order to prove the decidability of the *forward* fragment of XPath, allowing only axes navigating downward or rightward (NEXT-SIBLING and FOLLOWING-SIBLING). The two models of automata are incomparable: ATRA can express all regular tree languages while BUDA can express unary inclusion dependency properties (like "the data values labeled by a is a subset of those labeled by b"). In order to capture vertical XPath, the switch from top-down to bottom-up seems necessary to express formulas with upward navigation, and this also makes the decidability of the emptiness problem significantly more difficult. In [5], the decidability of the forward fragment of XPath is also obtained using a WSTS. This WSTS relies on a wqo over configurations. As our automaton model is bottom-up we have to work with *sets* of configurations and had to invoke the theory of ω^2*-quasi-orderings* [11, 13] in order to derive our wqo. The paper [2] contains a comprehensive survey of the known decidability results for various fragments of XPath, most of which cannot access data values. In the presence of data values, the notable new results since the publication of [2] are the downward [4] and the forward [5] fragments, as well as the fragment containing only the successor axis [3] (the latter closely related to first-order logic with two variables). As already mentioned, this paper solves one of the remaining open problems of [2].

Organization. In Section 3 we introduce the BUDA model and we show that it captures vertical XPath. The associated well-structured transition system and the outline of the proof to show the decidability of its reachability is in Section 4. Due to space limitations many proofs are omitted or only sketched. We refer the reader to [6, Chapter 7] for detailed proofs.

2 Preliminaries

Basic notation. Let $\wp(S)$ denote the set of subsets of S, and $\wp_{<\infty}(S)$ be the set of *finite* subsets of S. Let $\mathbb{N} = \{0, 1, 2, \dots\}$, $\mathbb{N}_+ = \{1, 2, 3, \dots\}$, and let $[n] := \{1, \dots, n\}$ for any $n \in \mathbb{N}_+$. We fix once and for all $\mathbb{D}$ to be any infinite domain of data values; for simplicity

in our examples we will consider $\mathbb{D} = \mathbb{N}$. In general we use letters $\mathbb{A}$, $\mathbb{B}$ for finite alphabets, the letter $\mathbb{D}$ for an infinite alphabet and the letters $\mathbb{E}$ and $\mathbb{F}$ for any kind of alphabet. By $\mathbb{E}^*$ we denote the set of finite sequences over $\mathbb{E}$, by $\mathbb{E}^+$ the set of finite sequences with at least one element over $\mathbb{E}$, and by $\mathbb{E}^\omega$ the set of infinite sequences over $\mathbb{E}$. We write ϵ for the empty sequence and '$\cdot$' as the concatenation operator between sequences. We write $|S|$ to denote the length of S (if S is a finite sequence), or its cardinality (if S is a set).

Regular languages. We make use of the many characterizations of regular languages over a finite alphabet $\mathbb{A}$. In particular, we use that a word language $\mathscr{L} \subseteq \mathbb{A}^*$ is regular iff there is a finite semigroup $(S, \cdot)$ with a distinguished subset $T \subseteq S$, and a semigroup homomorphism $h : \mathbb{A}^* \to S$ such that for all w with $|w| > 0$, $w \in \mathscr{L}$ iff $h(w) \in T$.

Unranked finite trees. By $\mathit{Trees}(\mathbb{E})$ we denote the set of finite ordered and unranked trees over an alphabet $\mathbb{E}$. We view each **position** in a tree as an element of $(\mathbb{N}_+)^*$. Formally, we define $\mathsf{POS} \subseteq \wp_{<\infty}((\mathbb{N}_+)^*)$ as the set of sets of finite tree positions, such that: $X \in \mathsf{POS}$ iff (a) $X \subseteq (\mathbb{N}_+)^*, |X| < \infty$; (b) X is prefix-closed; and (c) if $n{\cdot}(i+1) \in X$ for $i \in \mathbb{N}_+$, then $n{\cdot}i \in X$. A tree is then a mapping from a set of positions to letters of the alphabet $\mathit{Trees}(\mathbb{E}) := \{\mathbf{t} : P \to \mathbb{E} \mid P \in \mathsf{POS}\}$. By $\mathbf{t}|_x(y)$ we denote the subtree of $\mathbf{t}$ at position x: $\mathbf{t}|_x(y) = \mathbf{t}(x{\cdot}y)$. The root's position is the empty string and we denote it by 'ϵ'. The position of any other node in the tree is the concatenation of the position of its parent and the node's index in the ordered list of siblings. Along this work we use x, y, z, w, v as variables for positions, and i, j, k, l, m, n as variables for numbers. For example, $x{\cdot}i$ is a position which is not the root, that has x as parent position, and that has $i-1$ siblings to the left.

Given a tree $\mathbf{t} \in \mathit{Trees}(\mathbb{E})$, $\mathsf{pos}(\mathbf{t})$ denotes the domain of $\mathbf{t}$, which consists of the set of positions of the tree, and $\mathsf{alph}(\mathbf{t}) = \mathbb{E}$ denotes the alphabet of the tree. From now on, we informally refer by 'node' to a position x together with the value $\mathbf{t}(x)$.

Given two trees $\mathbf{t}_1 \in \mathit{Trees}(\mathbb{E})$, $\mathbf{t}_2 \in \mathit{Trees}(\mathbb{F})$ such that $\mathsf{pos}(\mathbf{t}_1) = \mathsf{pos}(\mathbf{t}_2) = P$, we define $\mathbf{t}_1 \otimes \mathbf{t}_2 : P \to (\mathbb{E}{\times}\mathbb{F})$ as $(\mathbf{t}_1 \otimes \mathbf{t}_2)(x) = (\mathbf{t}_1(x), \mathbf{t}_2(x))$.

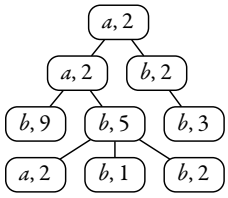

Figure 1 A data tree.

The set of **data trees** over a finite alphabet $\mathbb{A}$ and an infinite domain $\mathbb{D}$ is defined as $\mathit{Trees}(\mathbb{A}{\times}\mathbb{D})$. Note that every tree $\mathbf{t} \in \mathit{Trees}(\mathbb{A}{\times}\mathbb{D})$ can be decomposed into two trees $\mathbf{a} \in \mathit{Trees}(\mathbb{A})$ and $\mathbf{d} \in \mathit{Trees}(\mathbb{D})$ such that $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$. Figure 1 shows an example of a data tree. The notation for the set of data values used in a data tree is $\mathit{data}(\mathbf{a} \otimes \mathbf{d}) := \{\mathbf{d}(x) \mid x \in \mathsf{pos}(\mathbf{d})\}$. With an abuse of notation we write $\mathit{data}(X)$ to denote all the elements of $\mathbb{D}$ contained in X, for whatever object X may be.

XPath on data trees. Next we define vertical XPath, the fragment of XPath where no horizontal navigation is allowed. We actually consider an extension of XPath allowing the Kleene star on *any* path expression and we denote it by regXPath. Although we define this logic over data trees, our decidability result also holds for the class of XML documents through a standard reduction.

Vertical regXPath is a two-sorted language, with *path* expressions (that we write α, β, γ) and *node* expressions (φ, ψ, η). Path expressions are binary relations resulting from composing the child and parent relations (which are denoted respectively by $\downarrow$ and $\uparrow$), and node expressions. Node expressions are boolean formulas that test a property of a node, like for example, that is has a certain label, or that it has a child labeled a with the same data value as an ancestor labeled b, which is expressed by $\langle\downarrow[a] = \uparrow^*[b]\rangle$. We write $\mathsf{regXPath}(\mathfrak{V}, =)$ to denote this logic. A *formula* of $\mathsf{regXPath}(\mathfrak{V}, =)$ is either a node expression or a path expression of the logic. Its syntax and semantics are defined in Figure 2. As another example, we can select the nodes that have a descendant labeled b with two children also labeled by

$$\alpha,\beta \;::=\; o \mid \alpha[\varphi] \mid [\varphi]\alpha \mid \alpha\beta \mid \alpha\cup\beta \mid \alpha^* \qquad o\in\{\varepsilon,\downarrow,\uparrow\}\,,$$
$$\varphi,\psi \;::=\; a \mid \neg\varphi \mid \varphi\wedge\psi \mid \varphi\vee\psi \mid \langle\alpha\rangle \mid \langle\alpha=\beta\rangle \mid \langle\alpha\neq\beta\rangle \qquad a\in\mathbb{A}\,.$$

$$\begin{aligned}
[\![\downarrow]\!]^{\mathbf{t}} &= \{(x,x{\cdot}i) \mid x{\cdot}i\in\mathsf{pos}(\mathbf{t})\} & [\![\uparrow]\!]^{\mathbf{t}} &= \{(x{\cdot}i,x) \mid x{\cdot}i\in\mathsf{pos}(\mathbf{t})\}\\
[\![[\varphi]]\!]^{\mathbf{t}} &= \{(x,x) \in|\, x\in\mathsf{pos}(\mathbf{t}), x\in[\![\varphi]\!]^{\mathbf{t}}\} & [\![\alpha^*]\!]^{\mathbf{t}} &= \text{the reflexive transitive closure of } [\![\alpha]\!]^{\mathbf{t}}\\
[\![\varepsilon]\!]^{\mathbf{t}} &= \{(x,x) \mid x\in\mathsf{pos}(\mathbf{t})\} & [\![\alpha\beta]\!]^{\mathbf{t}} &= \{(x,z) \mid \text{ there exists } y \text{ such that}\\
& & & \quad (x,y)\in[\![\alpha]\!]^{\mathbf{t}}, (y,z)\in[\![\beta]\!]^{\mathbf{t}}\}\\
[\![\alpha\cup\beta]\!]^{\mathbf{t}} &= [\![\alpha]\!]^{\mathbf{t}}\cup[\![\beta]\!]^{\mathbf{t}} & [\![\langle\alpha\rangle]\!]^{\mathbf{t}} &= \{x\in\mathsf{pos}(\mathbf{t}) \mid \exists y.(x,y)\in[\![\alpha]\!]^{\mathbf{t}}\}\\
[\![a]\!]^{\mathbf{t}} &= \{x\in\mathsf{pos}(\mathbf{t}) \mid \mathbf{a}(x)=a\} & [\![\varphi\wedge\psi]\!]^{\mathbf{t}} &= [\![\varphi]\!]^{\mathbf{t}}\cap[\![\psi]\!]^{\mathbf{t}}\\
[\![\neg\varphi]\!]^{\mathbf{t}} &= \mathsf{pos}(\mathbf{t})\setminus[\![\varphi]\!]^{\mathbf{t}} & [\![\langle\alpha{\neq}\beta\rangle]\!]^{\mathbf{t}} &= \{x\in\mathsf{pos}(\mathbf{t}) \mid \exists y,z\ (x,y)\in[\![\alpha]\!]^{\mathbf{t}},\\
[\![\langle\alpha{=}\beta\rangle]\!]^{\mathbf{t}} &= \{x\in\mathsf{pos}(\mathbf{t}) \mid \exists y,z\ (x,y)\in[\![\alpha]\!]^{\mathbf{t}}, & & \quad (x,z)\in[\![\beta]\!]^{\mathbf{t}}, \mathbf{d}(y)\neq\mathbf{d}(z)\}\\
& \quad (x,z)\in[\![\beta]\!]^{\mathbf{t}}, \mathbf{d}(y)=\mathbf{d}(z)\}
\end{aligned}$$

Figure 2 The syntax of $\mathsf{XPath}(\mathfrak{V},=)$; and its semantics for a data tree $\mathbf{t}=\mathbf{a}\otimes\mathbf{d}$.

b with different data value by a formula $\varphi=\langle\downarrow^*[\,b\wedge\langle\downarrow[b]\neq\downarrow[b]\rangle\,]\rangle$. Given a tree $\mathbf{t}$ as in Figure 1, we have $[\![\varphi]\!]^{\mathbf{t}}=\{\varepsilon,1,12\}$.

The satisfiability problem for $\mathsf{regXPath}(\mathfrak{V},=)$ is the problem of, given a formula φ, wether there exists a data tree $\mathbf{t}$ such that $[\![\varphi]\!]^{\mathbf{t}}\neq\emptyset$. Our main result on XPath is the following.

▶ **Theorem 1.** *The satisfiability problem for* $\mathsf{regXPath}(\mathfrak{V},=)$ *is decidable.*

Well-structured transition systems. The proof of Theorem 1 relies on a translation to an automaton model defined in the next section whose emptiness problem is decidable. This is showed through methods from the theory of *well-structured transition systems*, or WSTS for short [8]. The emptiness test of our model of automata is obtained by interpreting its execution using a transition system compatible with some well-quasi-ordering (wqo). We reproduce here only the result of the theory of WSTS that we will need.

A quasi-order $\leq$ (*i.e.*, a reflexive and transitive relation) over a set S is said to be a **well-quasi-order** (wqo) iff for every infinite sequence $s_1\,s_2\,\cdots\in S^\omega$ there are two indices $i<j$ such that $s_i\leq s_j$. Given a wqo $(S,\leq)$ and $T\subseteq S$, we define the *downward closure* of T as $\downarrow T:=\{s\in S\mid\exists\, t\in T, s\leq t\}$ and T is **downward closed** if $\downarrow T=T$.

Given a transition system $(S,\to)$, and $T\subseteq S$ we define $Succ(T):=\{s\in S\mid\exists\, t\in T$ with $t\to s\}$, and $Succ^*$ as its reflexive-transitive closure. We say that $(S,\to)$ is *finitely branching* iff $Succ(\{s\})$ is finite for all $s\in S$. If $Succ(\{s\})$ is also effectively computable for all s, we say that $(S,\to)$ is **effective**.

We adapt some results and definitions of [8, §5] of what is there called *reflexive compatibility* (*i.e.*, compatibility in zero or one steps) to extend them to N-compatibility (*i.e.*, compatibility in at most N steps). Given a binary relation $R\subseteq S\times S$ and $K\subseteq S$, let us write $R^{\leq n}$ for $Id\cup R\cup R^2\cup\cdots\cup R^n$, where Id is the identity relation and R^i is the i-fold composition of R. Given $N\in\mathbb{N}_+$, a transition system $(S,\to)$ is **N-downward compatible** with respect to a wqo $(S,\leq)$ iff for every $s_1,s_2,s_1'\in S$ such that $s_1'\leq s_1$ and $s_1\to s_2$, there exists $s_2'\in S$ such that $s_2'\leq s_2$ and $s_1'\ (\to)^{\leq N}\ s_2'$. In some sense any behavior from the bigger element s_1 can be simulated by the smaller element s_1'. In truth, s_1' may need several transitions, but not more than N. Hereafter we use the term 'WSTS' to refer to any wqo and transition system N-downward compatible. A simple adaptation of [8, §5] yields the following proposition, what will be used to show decidability for BUDA.

▶ **Proposition 1.** *If* $(S,\leq)$ *is a* wqo *and* $(S,\to)$ *a transition system such that (1) it is* N-downwards compatible *for some fixed* N*, (2) it is effective, and (3)* $\leq$ *is decidable,* $V\subseteq S$ *is a recursive downward-closed set and,* $T\subseteq S$ *is a finite set; then the problem of whether there exist* $t\in T$ *and* $v\in V$ *such that* $t\to^* v$ *is decidable.*

In the above statement one must think of S as the configurations of an automaton, T as its initial configurations, $\rightarrow$ as the relation defined by the run, and $\leq$ as a suitable relation between configurations such that the set of accepting configurations V is downward-closed.

3 The automaton model

In this section we introduce the BUDA model. It is essentially a bottom-up tree automaton with one register to store a data value and an alternating control. We show that these automata are at least as expressive as vertical regXPath. In Section 4 we will show that their emptiness problem is decidable. Theorem 1 then follows immediately.

An automaton $\mathcal{A} \in$ BUDA that runs over data trees of $\mathit{Trees}(\mathbb{A} \times \mathbb{D})$ is defined as a tuple $\mathcal{A} = (\mathbb{A}, \mathbb{B}, Q, q_0, \delta_\epsilon, \delta_{up}, \mathcal{S}, h)$ where $\mathbb{A}$ is the finite alphabet of the tree, $\mathbb{B}$ is an internal finite alphabet of the automaton (whose purpose will be clear later), Q is a finite set of states, q_0 is the initial state, $\mathcal{S}$ is a finite semigroup, h is a semigroup homomorphism from $(\mathbb{A} \times \mathbb{B})^+$ to $\mathcal{S}$, δ_ϵ is the ϵ-transition function of $\mathcal{A}$, and δ_{up} is the *up*-transition function of $\mathcal{A}$.

δ_{up} is a partial function from states to formulas. For $q \in Q$, $\delta_{up}(q)$ is either undefined or a formula consisting in a disjunction of conjunctions of states. δ_ϵ is also a partial function from states to disjunctions of conjunctions of 'atoms' of one of the following forms:

$$p \mid \mathsf{guess}(p) \mid \mathsf{univ}(p) \mid \mathsf{store}(p) \mid \mathsf{eq} \mid \overline{\mathsf{eq}} \mid$$
$$\mid \langle\mu\rangle^{=} \mid \langle\mu\rangle^{\neq} \mid \overline{\langle\mu\rangle^{=}} \mid \overline{\langle\mu\rangle^{\neq}} \mid \mathsf{root} \mid \overline{\mathsf{root}} \mid \mathsf{leaf} \mid \overline{\mathsf{leaf}} \mid a \mid \bar{a} \mid b \mid \bar{b}$$

where $\mu \in \mathcal{S}$, $p \in Q$, $a \in \mathbb{A}$, $b \in \mathbb{B}$.

Before we present the precise semantics of our automaton model, here is the intuition. The automaton's control is nondeterministic and alternating, as reflected by the disjunctions and conjunctions in the formulæ specifying the transition functions. Hence, at any node several *threads* of the automaton run in parallel. Each thread consists of a state and a data value stored in the register. At every node of the tree, the automaton guesses a finite internal label of $\mathbb{B}$ and all threads can share access to this finite information. At any node of the tree, the automaton can perform some actions depending on the result of local tests.

We first describe the battery of tests the automata can perform. All these tests are explicitly closed under negation, denoted with the $\overline{\cdot}$ notation, and of course they are also closed under intersection and union using the alternating and nondeterministic control of the automata. The automata can test the label and internal label of the current node and also whether the current node is the root, a leaf or an internal node. The automata can test (in)equality of the current data value with the one stored in the register (eq and $\overline{\mathsf{eq}}$). Finally the automata can test the existence of some downward path, starting from the current node and leading to a node whose data value is (or is not) equal to the one currently stored in the register, such that the path satisfies some regular property on the labels. These properties are specified using the finite semigroup $\mathcal{S}$ and the morphism $h : (\mathbb{A} \times \mathbb{B})^+ \rightarrow \mathcal{S}$ over the words made of the label of the tree and the internal label. For example, $\langle\mu\rangle^{=}$ tests for the existence of a path that evaluates to μ via h, which starts at the current node and leads to a node whose data value matches the one currently stored in the register. Similarly, $\langle\mu\rangle^{\neq}$ tests that it leads to a data value different from the one currently in the register. Observe that we could have used finite automata, or regular expressions instead of finite semigroup homomorphisms. We take this approach because it simplifies the notation.

Based on the result of these tests, the automata can perform the following actions. They can change state, store the current data value in the register ($\mathsf{store}(p)$), or store an arbitrary data value nondeterministically chosen ($\mathsf{guess}(p)$). Finally, a transition can demand to start

a new thread in state p for every data value of the subtree with the operation $\mathsf{univ}(p)$. The automata can also decide to move up in the tree according to the *up*-transition.

Before we move on to the formal definition, we stress that the automaton model is not closed under complementation because its set of actions are not closed under complementation: guess is a form of existential quantification while univ is a form of universal quantification, but they are not dual. Actually, we will show in the journal version of this paper that adding any of their dual would yield undecidability of the model.

We now turn to the formal definition. A data tree $\mathbf{a} \otimes \mathbf{d} \in \mathit{Trees}(\mathbb{A} \times \mathbb{D})$ is accepted by $\mathcal{A}$ iff there exists an internal labeling $\mathbf{b} \in \mathit{Trees}(\mathbb{B})$ with $\mathsf{pos}(\mathbf{b}) = \mathsf{pos}(\mathbf{a} \otimes \mathbf{d})$ such that there is an accepting run on $\mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$. We focus now on the definition of a run.

A **configuration** of a BUDA $\mathcal{A}$ is a set $\mathcal{C}$ of threads, viewed as a finite subset of $Q \times \mathbb{D}$. A configuration $\mathcal{C}$ is said to be **initial** iff it is of the form $\{(q_0, e)\}$ for some $e \in \mathbb{D}$. A configuration $\mathcal{C}$ is **accepting** iff it is empty.

ϵ-transitions. Let $\mathbf{t} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$ and $x \in \mathsf{pos}(\mathbf{t})$. Given two configurations $\mathcal{C}$ and $\mathcal{C}'$ of $\mathcal{A}$, we say that there is an ϵ-transition of $\mathcal{A}$ at x between $\mathcal{C}$ and $\mathcal{C}'$, denoted $(x, \mathcal{C}) \rightarrowtail_\epsilon (x, \mathcal{C}')$ (assuming $\mathcal{A}$ and $\mathbf{t}$ are understood from the context) if the following holds: there is a thread with state q and with a data value d (*i.e.*, $(q, d) \in \mathcal{C}$) where $\delta_\epsilon(q) = \bigvee_{i \in I} \gamma_i$. Each γ_i is a conjunction of atoms, and there must be one $i \in I$ with $\gamma_i = \bigwedge_{j \in J} \alpha_j$ and $\mathcal{C}' = (\mathcal{C} \setminus \{(q, d)\}) \cup \hat{\mathcal{C}}$ such that the following holds for all $j \in J$:

- If α_j is one of the tests a, b, root, leaf, eq or its negations, it must be true with the obvious semantics as described above.
- If α_j is $\langle \mu \rangle^=$ then there is a downward path in $\mathbf{t}$ starting at x and ending at some descendant position y with $\mathbf{d}(y) = d$, such that the sequence of labels in $\mathbb{A} \times \mathbb{B}$ read while going from x to y along this path (including the endpoints) evaluates to μ via h. The case of $\langle \mu \rangle^\neq$ is treated similarly replacing $\mathbf{d}(y) = d$ by $\mathbf{d}(y) \neq d$. The tests $\overline{\langle \mu \rangle^=}$ and $\overline{\langle \mu \rangle^\neq}$ correspond to the negation of these tests.
- If α_j is p for some $p \in Q$, then $(p, d) \in \hat{\mathcal{C}}$,
- if α_j is $\mathsf{store}(p)$ then $(p, \mathbf{d}(x)) \in \hat{\mathcal{C}}$,
- if α_j is $\mathsf{guess}(p)$ then $(p, d') \in \hat{\mathcal{C}}$ for some $d' \in \mathbb{D}$,
- if α_j is $\mathsf{univ}(p)$, then for all $d' \in \mathit{data}(\mathbf{t}|_x)$, $(p, d') \in \hat{\mathcal{C}}$,
- nothing else is in $\hat{\mathcal{C}}$.

The **ϵ-closure** of a pair $(x, \mathcal{C})$ is defined as the reflexive transitive closure of $\rightarrowtail_\epsilon$, i.e. the set of configurations reachable from $(x, \mathcal{C})$ by a finite sequence of ϵ-transitions.

***up*-transitions.** We say that a configuration $\mathcal{C}$ is **moving** iff for all $(q, d) \in \mathcal{C}$, $\delta_{up}(q)$ is defined. Given two configurations $\mathcal{C}$ and $\mathcal{C}'$ of $\mathcal{A}$, we say that there is an *up*-transition of $\mathcal{A}$ between $\mathcal{C}$ and $\mathcal{C}'$, denoted $\mathcal{C} \rightarrowtail_{up} \mathcal{C}'$ (assuming $\mathcal{A}$ is understood from the context) if the following conditions hold:

- $\mathcal{C}$ is moving,
- for all $(q, d) \in \mathcal{C}$, if $\delta_{up}(q) = \bigvee_{i \in I} \bigwedge_{j \in J} p_{i,j}$ then there is $i \in I$ such that for all $j \in J$, $(p_{i,j}, d) \in \mathcal{C}'$,
- nothing else is in $\mathcal{C}'$.

▶ Remark. In the definition of the run the automaton behaves synchronously: all threads move up at the same time. This is only for convenience of presentation. Since all the threads are independent, one can also define a run in which each thread moves independently. This alternative definition would be equivalent to the current one.

Runs. We are now ready to define a **run** ρ of $\mathcal{A}$ on $\mathbf{t} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$. It is a function associating a configuration to any node x of $\mathbf{t}$ such that

1. for any leaf x of $\mathbf{t}$, $\rho(x) = \{(q_0, \mathbf{d}(x))\}$,
2. for any inner position x of $\mathbf{t}$ whose children are $x{\cdot}1, \ldots, x{\cdot}n$, then there are configurations $\mathcal{C}'_1, \cdots, \mathcal{C}'_n$ and $\mathcal{C}''_1, \cdots, \mathcal{C}''_n$ such that for all $i \in [n]$, $(x{\cdot}i, \mathcal{C}''_i)$ is in the ϵ-closure of $(x{\cdot}i, \rho(x{\cdot}i))$, $\mathcal{C}''_i \rightarrowtail_{up} \mathcal{C}'_i$, and $\rho(x) = \bigcup_{i \in [n]} \mathcal{C}'_i$.

The run ρ is **accepting** if moreover at the root (i.e., for the position ϵ), the ϵ-closure of $\rho(\epsilon)$ contains an accepting configuration.

BUDA and vertical regXPath. Given a formula η of $\mathsf{regXPath}(\mathfrak{V}, =)$, we say that a BUDA $\mathcal{A}$ is *equivalent* to η if a data tree $\mathbf{t}$ is accepted by $\mathcal{A}$ iff $[\![\eta]\!]^{\mathbf{t}} \neq \emptyset$.

▶ **Proposition 2.** *For every $\eta \in \mathsf{regXPath}(\mathfrak{V}, =)$ there exists an equivalent $\mathcal{A} \in$ BUDA computable from η.*

Proof idea. It is easy to simulate any positive test $\langle \alpha = \beta \rangle$ or $\langle \alpha \neq \beta \rangle$ of vertical regXPath by a BUDA using guess, $\langle \mu \rangle^{=}$ and $\langle \mu \rangle^{\neq}$. For example, consider the property $\langle \downarrow_* [a] \neq \uparrow\downarrow [b] \rangle$, which states that there is a descendant labeled a with a different data value than a sibling labeled b. A BUDA automaton can test this property as follows.

1. It guesses a data value d and stores it in the register.
2. It tests that d can be reached by $\downarrow_* [a]$ with a test $\langle \mu \rangle^{=}$ for a suitable μ.
3. It moves up to its parent.
4. It tests that a different value than d can be reached in one of its children labeled with b, using the test $\langle \mu \rangle^{\neq}$ for a suitable μ.

The simulation of negative tests ($\neg\langle \alpha = \beta \rangle$ or $\neg\langle \alpha \neq \beta \rangle$) is more tedious as BUDA is not closed under complementation. Nevertheless, the automaton has enough universal quantifications (in the operations univ, $\overline{\langle \mu \rangle^{=}}$ and $\overline{\langle \mu \rangle^{\neq}}$) in order to do the job. Consider for example the formula $\neg\langle \uparrow^* [b] \downarrow [a] = \downarrow_* [c] \rangle$, that states that no data value is shared between a descendant labeled c and an a-child of a b-ancestor. The automaton behaves as follows.

1. It creates one thread in state q for every data value in the subtree, using $\mathsf{univ}(q)$.
2. q tests whether the data value of the register is reachable by $\downarrow_* [c]$, using a test $\langle \mu \rangle^{=}$. If it is, it changes to state p.
3. p moves up towards the root, and each time it finds a b, it tests that the currently stored data value cannot be reached by $\downarrow [a]$. This is done with a test of the kind $\overline{\langle \mu \rangle^{=}}$. ◀

Therefore, in order to conclude the proof of Theorem 1, it remains to show that the emptiness problem of BUDA is decidable. This is the goal of Section 4.

Automata normal form. We now present a normal form of BUDA, removing all the redundancy in its definition. This normal form simplifies the technical details in the proof of decidability presented in the next section.

(NF1) The semigroup $\mathcal{S}$ and morphism h have the following property. For all $w \in (\mathbb{A} \times \mathbb{B})^+$ and $c \in \mathbb{A} \times \mathbb{B}$, $h(w) = h(c)$ iff $w = c$.

(NF2) In the definition of δ_{up} of $\mathcal{A}$, there is exactly one disjunct that contains exactly one conjunct. That is, for all $q \in Q$, $\delta_{\mathrm{up}}(q)$ is undefined or $\delta_{\mathrm{up}}(q) = p$ for some $p \in Q$.

(NF3) For all $q \in Q$, $\delta_\epsilon(q)$ is defined either as an atom, as $p \wedge p'$ or as $p \vee p'$ for some $p, p' \in Q$.

(NF4) For all $q \in Q$, $\delta_\epsilon(q)$ does not contain tests for labels (a, $\bar{a}$, b, $\bar{b}$), eq, $\overline{\mathsf{eq}}$, store, leaf or $\overline{\mathsf{leaf}}$.

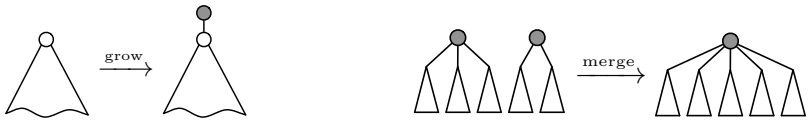

Figure 3 The *grow* and *merge* operations.

An automaton $\mathcal{A} \in$ BUDA is said to be in **normal form** if it satisfies (NF1), (NF2), (NF3) and (NF4). Notice that once (NF1) holds, then any test concerning a label (a, $\bar{a}$, b, or $\bar{b}$) can be simulated using tests of the form $\langle \mu \rangle$ for some appropriate μ. Using similar ideas, it is not hard to check that:

▶ **Proposition 3.** *For any $\mathcal{A} \in$ BUDA, there is an equivalent $\mathcal{A}' \in$ BUDA in normal form that can be effectively obtained.*

4 The emptiness problem for BUDA

The goal of this section is to show:

▶ **Theorem 2.** *The emptiness problem for* BUDA *is decidable.*

In order to achieve this, we associate with each BUDA a WSTS that simulates its runs. The transition system works on sets of *abstract configurations*. Given an automaton, an abstract configuration consists of all the information of the run that is necessary to maintain at the root of a given subtree in order to continue the simulation of the automaton from there. The aforesaid transition system works with *sets* of such abstract configurations in order to capture the bottom-up behavior of the automaton on unranked trees. The transition relation of the WSTS essentially corresponds to the transitions of the automaton except for the *up*-transition. An *up*-transition of the automaton is simulated by a succession of two types of transitions of the WSTS, called *grow* and *merge*. The object of doing this is to avoid having transitions that take an unbounded number of arguments (as the *up* relation in the run of the automaton does). The *grow* transition adds a node on top of the current root, and the *merge* transition identifies the roots of two abstract configurations. Intuitively, these transitions correspond to the operations on trees of Figure 3. This is necessary because we do not know in advance the arity of the tree and therefore the transition system has to build one subtree at a time. We then exhibit a wqo on abstract configurations and show that the transition system is N-downward compatible with respect to this wqo for some N that depends on the automaton. Decidability will then follow from Proposition 1.

In the sequel we implicitly assume that all our BUDA are in normal form.

4.1 Abstract configurations

Given a BUDA $\mathcal{A} = (\mathbb{A}, \mathbb{B}, Q, q_0, \delta_\epsilon, \delta_{up}, \mathcal{S}, h)$, we start the definition of its associated WSTS by defining its universe: finite sets of abstract configurations of $\mathcal{A}$.

An **abstract configuration** of $\mathcal{A}$ is a tuple (Δ, Γ, r, m) where r and m are either true or false, Δ is a finite subset of $Q \times \mathbb{D}$ and Γ is a finite subset of $\mathcal{S} \times \mathbb{D}$ such that

$$\Gamma \text{ contains exactly one pair of the form } (h(c), d) \text{ with } c \in \mathbb{A} \times \mathbb{B}. \qquad (\star)$$

This unique element of $\mathbb{A} \times \mathbb{B}$ is denoted as the **label** of the abstract configuration and the unique associated data value is denoted as the **data value** of the abstract configuration.

Intuitively r says whether the current node should be treated as the root or not, m says whether we are in a phase of *merging configurations* or not (a notion that will become clear when we introduce our transition system later on), Δ is the set of ongoing threads (corresponding to the configuration of the automaton) and a pair $(\mu, d) \in \Gamma$ simulates the existence of a downward path evaluating to μ and whose last element carries the datum d.

In the sequel we will use the following notation: $\Delta(d) = \{q \mid (q,d) \in \Delta\}$, $\Gamma(d) = \{\mu \mid (\mu,d) \in \Gamma\}$, $\Delta(q) = \{d \mid (q,d) \in \Delta\}$, $\Gamma(\mu) = \{d \mid (\mu,d) \in \Gamma\}$. We also use the notation $\Delta \otimes \Gamma : \mathbb{D} \to \wp(Q) \times \wp(\mathcal{S})$ with $(\Delta \otimes \Gamma)(d) = (\Delta(d), \Gamma(d))$. Given a data value d and an abstract configuration θ, $(\Delta \otimes \Gamma)(d)$ is also denoted as the **type of** d **in** θ. We use the letter θ to denote an abstract configuration and we write AC to denote the set of all abstract configurations. Similarly, we use Θ to denote a finite set of abstract configurations and $\wp_{<\infty}(\text{AC})$ for the set of finite sets of abstract configurations.

An abstract configuration $\theta = (\Delta, \Gamma, r, m)$ is said to be **initial** if it corresponds to a leaf node, i.e., is such that $\Delta = \{(q_0, d)\}$ and $\Gamma = \{(h(a), d)\}$ for some $d \in \mathbb{D}$ and $a \in \mathbb{A} \times \mathbb{B}$. It is said to be **accepting** if Δ is empty and r is *true*.

Two configurations θ_1 and θ_2 are said to be **equivalent** if there is a bijection $f : \mathbb{D} \to \mathbb{D}$ such that $f(\theta_1) = \theta_2$ (with some abuse of notation). In this case we note $\theta_1 \sim \theta_2$.

Finally, we write Θ_I to denote the set of all initial abstract configurations modulo $\sim$ (i.e., a set containing at most one element for each $\sim$ equivalence class). Note that Θ_I is *finite* and effective. A set of abstract configurations is said to be **accepting** iff it contains an accepting abstract configuration.

4.2 Well-quasi-orders

We now equip $\wp_{<\infty}(\text{AC})$ with a well-quasi-order $(\wp_{<\infty}(\text{AC}), \leq_{min})$. The order $\leq_{min}$ builds upon a wqo $(\text{AC}, \precsim)$ over abstract configurations. Let us define these orderings precisely.

The **profile of an abstract configuration** $\theta = (\Delta, \Gamma, r, m)$, denoted by $\mathit{profile}(\theta)$, is $\mathit{profile}(\theta) = (\ A_0,\ A_1,\ r,\ m\ )$ with $A_i = \{(S, \chi) \in \wp(Q) \times \wp(\mathcal{S}) : |(\Delta \otimes \Gamma)^{-1}(S, \chi)| = i\}$.

We first define the quasi-order $\preceq$ over abstract configurations, and then we define the order $(\text{AC}, \precsim)$ as $(\text{AC}, \preceq)$ modulo $\sim$. Given two abstract configurations $\theta_1 = (\Delta_1, \Gamma_1, r_1, m_1)$ and $\theta_2 = (\Delta_2, \Gamma_2, r_2, m_2)$, we denote by $\theta_1 \preceq \theta_2$ the fact that

- $\mathit{profile}(\theta_1) = \mathit{profile}(\theta_2)$, and
- $(\Delta_1 \otimes \Gamma_1) \subseteq (\Delta_2 \otimes \Gamma_2)$.

▶ Remark. Notice that due to condition $(\star)$, $\theta_1 \preceq \theta_2$ implies that θ_1 and θ_2 have the same label and same data value.

We now define $\precsim$ as: $\theta_1 \precsim \theta_2$ iff $\theta_1' \preceq \theta_2$ for some $\theta_1' \sim \theta_1$.

We are now ready to define our wqo over $\wp_{<\infty}(\text{AC})$. Given Θ_1 and Θ_2 in $\wp_{<\infty}(\text{AC})$ we define $\leq_{min}$ as: $\Theta_1 \leq_{min} \Theta_2$ iff for all $\theta_2 \in \Theta_2$ there is $\theta_1 \in \Theta_1$ such that $\theta_1 \precsim \theta_2$. That is, every element from Θ_2 is *minorized* by an element of Θ_1.

The following is a key observation:

▶ **Lemma 3.** $(\wp_{<\infty}(\text{AC}), \leq_{min})$ *is a* wqo.

Finally, the following obvious lemma will be necessary to apply Proposition 1.

▶ **Lemma 4.** $\{\Theta \in \wp_{<\infty}(\text{AC}) \mid \Theta$ *is accepting*$\}$ *is downward closed for* $(\wp_{<\infty}(\text{AC}), \leq_{min})$.

4.3 Transition system

We now equip $\wp_{<\infty}(\text{AC})$ with a transition relation $\Rightarrow$. This transition relation is built upon a transition relation $\to$ over AC.

Let us first define $\to$ over AC. It is specified so that it reflects the transitions of $\mathcal{A}$. For each possible test of the automaton there is a transition that simulates this test by using the information contained in Γ, and removes the corresponding in Δ. And for every operation store, guess, univ there is a transition that modifies Δ. We call the ϵ-transitions of $\to$, that we note $\to_\epsilon$. On the other hand, any *up* transition of $\mathcal{A}$ is decomposed into transitions $\xrightarrow{\text{merge}}$ and $\xrightarrow{\text{grow}}$ that modify not only Δ but also Γ accordingly.

We start with ϵ-transitions. Given two abstract configurations $\theta_1 = (\Delta_1, \Gamma_1, r_1, m_1)$ and $\theta_2 = (\Delta_2, \Gamma_2, r_2, m_2)$, we say that $\theta_1 \to_\epsilon \theta_2$ if $m_1 = m_2 = \mathit{false}$ (the merge information is used for simulating an *up*-transition as will be explained later), $r_2 = r_1$ (whether the current node is the root or not should not change), θ_1 and θ_2 have the same label and data value, $\Gamma_2 = \Gamma_1$ (the tree is not affected by an ϵ-transition) and, furthermore, one of the following conditions holds:

1. $\theta_1 \xrightarrow{\mathsf{univ}} \theta_2$. This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = \mathsf{univ}(p)$ for some $p, q \in Q$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\}) \cup \{(p, e) : \exists \mu \,.\, (\mu, e) \in \Gamma_1\}$.
2. $\theta_1 \xrightarrow{\mathsf{guess}} \theta_2$. This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = \mathsf{guess}(p)$ for some $p, q \in Q$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\}) \cup \{(p, d')\}$ for some $d' \in \mathbb{D}$.
3. $\theta_1 \xrightarrow{\langle\mu\rangle^=} \theta_2$ (resp. $\theta_1 \xrightarrow{\langle\mu\rangle^\neq} \theta_2$). This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = \langle\mu\rangle^=$ (resp. $\delta_\epsilon(q) = \langle\mu\rangle^\neq$) for some $q \in Q$, $\mu \in \mathcal{S}$, and $\mu \in \Gamma_1(d)$ (resp. there exists $e \in \mathbb{D}$, $e \neq d$ such that $\mu \in \Gamma_1(e)$). In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\})$. The negation of these tests $\xrightarrow{\overline{\langle\mu\rangle^=}}$ and $\xrightarrow{\overline{\langle\mu\rangle^\neq}}$ are defined in a similar way.
4. $\theta_1 \xrightarrow{\mathsf{root}} \theta_2$ (resp. $\theta_1 \xrightarrow{\overline{\mathsf{root}}} \theta_2$). This transition can happen if there is $(q, d) \in \Delta$ with $\delta_\epsilon(q) = \mathsf{root}$ and $r_1 = \mathit{true}$ (resp. $\delta_\epsilon(q) = \overline{\mathsf{root}}$ and $r_1 = \mathit{false}$). In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\})$.
5. $\theta_1 \xrightarrow{\wedge} \theta_2$. This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = p \wedge p'$ for some $p, p', q \in Q$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\}) \cup \{(p, d), (p', d)\}$.
6. $\theta_1 \xrightarrow{\vee} \theta_2$. This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = p \vee p'$ for some $p, p', q \in Q$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\}) \cup A$, for $A = \{(p, d)\}$ or $A = \{(p', d)\}$.

Note that by (NF3) and (NF4) for every possible definition of $\delta_\epsilon(q)$ there is one transition that simulates it. To simulate δ_{up}, it turns out that we will need one extra ϵ-transition that makes our trees fatter. This transition assumes the same constraints as for $\to_\epsilon$ except that we no longer have $\Gamma_2 = \Gamma_1$. The idea is that this transition corresponds to duplicating all the immediate subtrees of the root. For example, if the root has $\mathbf{t}_1$ and $\mathbf{t}_2$ as subtrees, consider the operation of now having $\mathbf{t}_1$ $\mathbf{t}_2$ $\mathbf{t}'_1$ $\mathbf{t}'_2$ as subtrees, where $\mathbf{t}'_1$ and $\mathbf{t}'_2$ are identical to $\mathbf{t}_1$ and $\mathbf{t}_2$ except for one data value with profile (S, χ) that in $\mathbf{t}'_1$ and $\mathbf{t}'_2$ is replaced by a fresh data value. Here is the definition that follows this idea in terms of our transition system. We say that $\theta_1 \xrightarrow{\mathrm{inc}(S,\chi)} \theta_2$ for some pair $(S, \chi) \in \wp(Q) \times \wp(\mathcal{S})$ if $|(\Delta_1 \otimes \Gamma_1)^{-1}(S, \chi)| \geq 1$ and, either $\chi = \emptyset$ or $|(\Gamma_1)^{-1}(\chi)| \geq 2$. Then θ_2 is such that $\mathit{data}(\theta_2) = \mathit{data}(\theta_1) \cup \{e\}$ for some $e \notin \mathit{data}(\theta_1)$, $(\Delta_2 \otimes \Gamma_2)(e) = (S, \chi)$, and for all $d \neq e$, $(\Delta_2 \otimes \Gamma_2)(d) = (\Delta_1 \otimes \Gamma_1)(d)$. Observe that $\xrightarrow{\mathrm{inc}(S,\chi)}$ does not change the truth value of any test. Indeed, any test ($\langle\mu\rangle^=$, $\overline{\langle\mu\rangle^=}$, eq, etc.) that is true in θ, continues to be true after a $\xrightarrow{\mathrm{inc}(S,\chi)}$ transition, and vice-versa.

We define the transitions of the WSTS that correspond to *up*-transitions in the automaton. We split them into two phases: adding a new root symbol and merging the roots.

1. $\theta_1 \xrightarrow{\text{grow}} \theta_2$. Given two abstract configurations θ_1 and θ_2 as above, we say $\theta_1 \xrightarrow{\text{grow}} \theta_2$ if $r_1 = m_1 = \mathit{false}$, and for all $(q, d) \in \Delta_1$, $\delta_{up}(q)$ is defined and θ_2 is such that $\Delta_1 \rightarrowtail_{up} \Delta_2$,

and $\Gamma_2 = \{(\mu', e) : (\mu, e) \in \Gamma_1, \mu' = h(c)\cdot\mu\} \cup \{(h(c), d)\}$, for some $c \in \mathbb{A} \times \mathbb{B}$ and $d \in \mathbb{D}$. Notice that c and d are then the label and data value of θ_2. As a consequence of the normal form (NF1) of the semigroup, this operation preserves property $(\star)$.

2. $\theta_1, \theta_2 \xrightarrow{\text{merge}} \theta_0$. Given 3 abstract configurations $\theta_1 = (\Delta_1, \Gamma_1, r_1, m_1)$, $\theta_2 = (\Delta_2, \Gamma_2, r_2, m_2)$, $\theta_0 = (\Delta_0, \Gamma_0, r_0, m_0)$ we define $\theta_1, \theta_2 \xrightarrow{\text{merge}} \theta_0$ if they all have the same label and data value, $m_1 = m_2 = \mathit{true}$, $r_1 = r_2 = r_0$, $\Delta_0 = \Delta_1 \cup \Delta_2$, and $\Gamma_0 = \Gamma_1 \cup \Gamma_2$. Notice that this operation preserves property $(\star)$.

▶ Remark. $\xrightarrow{\text{inc}(S,\chi)}$ can be seen as a kind of $\xrightarrow{\text{merge}}$ which preserves the truth of tests.

We are now ready to define the transition relation over $\wp_{<\infty}(\text{AC})$.

▶ **Definition 5.** We define that $\Theta_1 \Rightarrow \Theta_0$ if one the following conditions holds:

1. There is $\theta_1 \in \Theta_1$ and $\theta_1' \sim \theta_1$ such that $\theta_1' \rightarrow_\epsilon \theta_0$ or $\theta_1' \xrightarrow{\text{inc}(S,\chi)} \theta_0$ or $\theta_1' \xrightarrow{\text{grow}} \theta_0$, for some θ_0, χ, and $\Theta_0 = \Theta_1 \cup \{\theta_0\}$.
2. There are $\theta_1, \theta_2 \in \Theta_1$ and $\theta_1' \sim \theta_1$, $\theta_2' \sim \theta_2$ such that $\theta_1', \theta_2' \xrightarrow{\text{merge}} \theta_0$ for some θ_0, and $\Theta_0 = \Theta_1 \cup \{\theta_0\}$.

In the definition of the transition system, the m flag is simply used to constrain the transition system to have all its $\xrightarrow{\text{merge}}$ operations right after $\xrightarrow{\text{grow}}$ and before any $\rightarrow_\epsilon$. Thus, if we take a derivation and examine the kind of $\rightarrow$ transitions that originated each $\Rightarrow$ transition, we obtain a word described by the following regular expression

$$\big((\rightarrow_\epsilon \mid \xrightarrow{\text{inc}(S,\chi)})^* \xrightarrow{\text{grow}} (\xrightarrow{\text{merge}})^*\big)^* (\rightarrow_\epsilon \mid \xrightarrow{\text{inc}(S,\chi)})^* \,. \qquad (\dagger)$$

4.4 Compatibility

We now show that all the previous definitions were chosen appropriately and that the transition system defined in Section 4.3 is compatible with the wqo defined in Section 4.2. The proof of this result is very technical and consists in a case analysis over each possible kind of transition. In this proof, the operation $\xrightarrow{\text{inc}(S,\chi)}$ becomes crucial to show that the downwards compatibility can always be done in a bounded amount of N steps. The detailed proof will appear in the journal version of this paper.

▶ **Proposition 4.** *The transition system* $(\wp_{<\infty}(\text{AC}), \Rightarrow)$ *is* $\mathbf{N}$*-downward compatible with respect to* $(\wp_{<\infty}(\text{AC}), \leq_{min})$*, for* $\mathbf{N} := 2 \cdot (|\mathcal{S}| \cdot |Q|)^2 + 1$.

Let $\equiv$ be the equivalence relation over $\wp_{<\infty}(\text{AC})$ such that $\Theta \equiv \Theta'$ iff $\Theta \leq_{min} \Theta'$ and $\Theta' \leq_{min} \Theta$. Given a BUDA $\mathcal{A}$, the WSTS $(\wp_{<\infty}(\text{AC})/\equiv, \Rightarrow, \leq_{min})$ as built in the previous section is called *the* WSTS *associated with* $\mathcal{A}$. From Proposition 1 and 4 we obtain:

▶ **Corollary 6.** *Given a* BUDA $\mathcal{A}$*, it is decidable whether the* WSTS *associated with* $\mathcal{A}$ *can reach an accepting abstract configuration from its initial abstract configuration.*

As shown next, this implies the decidability for the emptiness problem for BUDA.

4.5 From BUDA to its abstract configurations

As expected, the WSTS associated with a BUDA $\mathcal{A}$ reflects its behavior. That is, reachability of one corresponds exactly to accessibility of the other. One direction is easy as the transition system can easily simulate $\mathcal{A}$. The other direction requires more care. As evidenced in $(\dagger)$, the WSTS may perform a $\xrightarrow{\text{inc}(S,\chi)}$ transition anytime. However, BUDA can only make the tree grow in width when moving up in the tree. This issue is solved by showing that all other transitions commute with $\xrightarrow{\text{inc}(S,\chi)}$. Finally we obtain the following.

▶ **Proposition 5.** *Let $\mathcal{A}$ be a* BUDA. *Let $\mathcal{W}$ be the* WSTS *associated with $\mathcal{A}$. Then $\mathcal{A}$ has an accepting run iff $\mathcal{W}$ can reach an accepting set of abstract configurations from the initial set of abstract configurations.*

Hence, combining Proposition 5 and Corollary 6 we prove Theorem 2.

5 Concluding remarks

We have exhibited a decidable class of automata over data trees. This automaton model is powerful enough to code node expressions of $\mathsf{regXPath}(\mathfrak{V}, =)$. Therefore, since these expressions are closed under negation, we have shown decidability of the satisfiability, inclusion and equivalence problems for node expressions of $\mathsf{regXPath}(\mathfrak{V}, =)$.

Notice that if our result implies also the decidability of the emptiness problem for **path** expressions of $\mathsf{regXPath}(\mathfrak{V}, =)$, those being not closed under complementation it is not clear that their inclusion or equivalence problem remains decidable.

Our decision algorithm relies heavily on the fact that we work with **unranked** data trees. As already shown in [7] without this assumption $\mathsf{XPath}(\mathfrak{V}, =)$ would be undecidable. In particular if we further impose the presence of a DTD, $\mathsf{XPath}(\mathfrak{V}, =)$ becomes undecidable.

Finally we remark that our decision algorithm is not primitive recursive. But as shown in [7] there cannot be a primitive recursive decision algorithm for $\mathsf{XPath}(\mathfrak{V}, =)$.

References

1 Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *Journal of the ACM*, 55(2):1–79, 2008.

2 Michael Benedikt and Christoph Koch. XPath leashed. *ACM Computing Surveys*, 41(1), 2008.

3 Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3):1–48, 2009.

4 Diego Figueira. Satisfiability of downward XPath with data equality tests. In *ACM Symposium on Principles of Database Systems (PODS'09)*, 2009.

5 Diego Figueira. Forward-XPath and extended register automata on data-trees. In *International Conference on Database Theory (ICDT'10)*, 2010.

6 Diego Figueira. *Reasoning on words and trees with data*. PhD thesis, ÉNS de Cachan, 2010. Available at `http://www.lsv.ens-cachan.fr/~figueira/phd/`.

7 Diego Figueira and Luc Segoufin. Future-looking logics on data words and trees. In *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS'09)*, 2009.

8 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

9 Floris Geerts and Wenfei Fan. Satisfiability of XPath queries with sibling axes. In *Intl. Symp. on Database Programming Languages (DBPL'05)*, 2005.

10 Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005.

11 Petr Jančar. A note on well quasi-orderings for powersets. *Information Processing Letters*, 72(5-6):155–160, 1999.

12 Marcin Jurdziński and Ranko Lazić. Alternation-free modal mu-calculus for data trees. In *Logic in Computer Science (LICS'07)*, 2007.

13 Alberto Marcone. Foundations of bqo theory. *Transactions of the American Mathematical Society*, 345:641–660, 1994.