Alternating Automata on Data Trees and XPath Satisfiability

MARCIN JURDZIŃSKI and RANKO LAZIĆ, University of Warwick

A data tree is an unranked ordered tree whose every node is labeled by a letter from a finite alphabet and an element ("datum") from an infinite set, where the latter can only be compared for equality. The article considers alternating automata on data trees that can move downward and rightward, and have one register for storing data. The main results are that nonemptiness over finite data trees is decidable but not primitive recursive, and that nonemptiness of safety automata is decidable but not elementary. The proofs use nondeterministic tree automata with faulty counters. Allowing upward moves, leftward moves, or two registers, each causes undecidability. As corollaries, decidability is obtained for two data-sensitive fragments of the XPath query language.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Decision problems*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata*; H.2.3 [**Database Management**]: Languages—*Query languages*

General Terms: Algorithms, Verification

ACM Reference Format:
Jurdziński, M., and Lazić, R. 2011. Alternating automata on data trees and XPath satisfiabilty. ACM Trans. Comput. Logic 12, 3, Article 19 (May 2011), 21 pages.
DOI = 10.1145/1929954.1929956 http://doi.acm.org/10.1145/1929954.1929956

1. INTRODUCTION

Context. Logics and automata for words and trees over finite alphabets are relatively well understood. Motivated partly by the search for automated reasoning techniques for XML and the need for formal verification and synthesis of infinite-state systems, there is an active and broad research program on logics and automata for words and trees that have richer structure.

Initial progress made on reasoning about data words and data trees is summarized in the survey by Segoufin [2006]. A data word is a word over $\Sigma \times \mathcal{D}$, where Σ is a finite alphabet, and $\mathcal{D}$ is an infinite set (domain) whose elements (data) can only be compared for equality. Similarly, a data tree is a tree (countable, unranked and ordered) whose every node is labeled by a pair in $\Sigma \times \mathcal{D}$.

First-order logic for data words was considered by Bojańczyk et al. [2006], and related automata were studied further by Björklund and Schwentick [2007]. The logic has variables that range over word positions ($\{0, \ldots, l-1\}$ or $\mathbb{N}$), a unary predicate for each letter from the finite alphabet, and a binary predicate $x \sim y$ that denotes equality

---

This article is a revised and extended version of Jurdziński and Lazić [2007].
R. Lazić was supported by a grant from the Intel Corporation.
Authors' address: M. Jurdziński and R. Lazić, University of Warwick, Gibbet Hill Road, Coventry CV4 7AL, UK, email: {R.S.Lasic, Marcin.Jurdzinski}@warwick.ac.uk.

DOI 10.1145/1929954.1929956 http://doi.acm.org/10.1145/1929954.1929956

of data labels. $\mathrm{FO}^2(+1, <, \sim)$ denotes such a logic with two variables and binary predicates $x + 1 = y$ and $x < y$. Over finite and over infinite data words, satisfiability for $\mathrm{FO}^2(+1, <, \sim)$ was shown decidable and at least as hard as nonemptiness of vector addition automata. Whether the latter problem is elementary has been open for many years. Extending the logic by one more variable causes undecidability.

Over data trees, $\mathrm{FO}^2(+1, <, \sim)$ denotes a similar first-order logic with two variables. The variables range over tree nodes, $+1$ stands for two predicates "child" and "next sibling", and $<$ stands for two predicates "descendant" and "younger sibling". Complexity of satisfiability over finite data trees was studied by Bojańczyk et al. [2009]. For $\mathrm{FO}^2(+1, \sim)$, it was shown to be in 3NExpTime, but for $\mathrm{FO}^2(+1, <, \sim)$, to be at least as hard as nonemptiness of vector addition tree automata. Decidability of the latter is an open question, and it is equivalent to decidability of multiplicative exponential linear logic [deGroote et al. 2004]. However, Björklund and Bojańczyk [2007] showed that $\mathrm{FO}^2(+1, <, \sim)$ over finite data trees of bounded depth is decidable.

XPath [Clark and DeRose 1999] is a prominent query language for XML documents [Bray et al. 1998]. The most basic static analysis problem for XPath, with a variety of applications, is satisfiability in the presence of DTDs. In the two extensive articles on its complexity [Benedikt et al. 2008; Geerts and Fan 2005], the only decidability result that allows negation and data (i.e., equality comparisons between attribute values) does not allow axes which are recursive (such as "self or descendant") or between siblings. By representing XML documents as data trees and translating from XPath to $\mathrm{FO}^2(+1, \sim)$, Bojańczyk et al. [2009] obtained a decidable fragment with negation, data and all nonrecursive axes. Another fragment of XPath was considered by Hallé et al. [2006], but it lacks concatenation, recursive axes and sibling axes. A recent advance of Figueira [2009] shows ExpTime-completeness for full downward XPath, but with restricted DTDs.

An alternative approach to reasoning about data words is based on automata with registers [Kaminski and Francez 1994]. A register is used for storing a datum for later equality comparisons. Nonemptiness of one-way nondeterministic register automata over finite data words has relatively low complexity: NP-complete [Sakamoto and Ikeda 2000] or PSpace-complete [Demri and Lazić 2009], depending on technical details of their definition. Unfortunately, such automata fail to provide a satisfactory notion of regular language of finite data words, as they are not closed under complement [Kaminski and Francez 1994] and their nonuniversality is undecidable [Neven et al. 2004]. To overcome those limitations, one-way alternating automata with 1 register were proposed by Demri and Lazić [2009]: they are closed under Boolean operations, their nonemptiness over finite data words is decidable, and future-time fragments of temporal logics such as LTL or the modal μ-calculus extended by 1 register are easily translatable to such automata. However, the nonemptiness problem over finite data words turned out to be not primitive recursive. Moreover, already with weak acceptance [Muller et al. 1986] and thus also with Büchi or co-Büchi acceptance, nonemptiness over infinite data words is undecidable (more precisely, co-r.e.-hard). When the automata are restricted to those which recognise safety properties [Alpern and Schneider 1987] over infinite data words, nonemptiness was shown to be ExpSpace-complete, and inclusion to be decidable but not primitive recursive [Lazić 2006].

Contribution. This article addresses one of the research directions proposed by Segoufin [2006]: investigating modal logics and automata with registers on data trees. Nondeterministic automata with registers that can be nondeterministically reassigned on finite binary data trees were recently studied by Kaminski and Tan [2008]: top-down and bottom-up variants recognise the same languages, and nonemptiness is decidable.

However, they inherit the drawbacks of one-way nondeterministic register automata on data words: lack of closure under complement and undecidability of nonuniversality.

We consider alternating automata that have 1 register and are forward, that is, can move downward and rightward over tree nodes: for short, ATRA_1. They are closed under Boolean operations, and we show that their nonemptiness over finite data trees is decidable. Moreover, forward fragments of CTL and the modal μ-calculus with 1 register are easily translatable to ATRA_1 [Jurdziński and Lazić 2007]. The expressiveness of ATRA_1 is incomparable to those of $\mathrm{FO}^2(+1, \sim)$ and the automata of Kaminski and Tan [2008]: for example, the latter two formalisms but not ATRA_1 can check whether some two leaves have equal data, and the opposite is true of checking whether each node's datum is fresh, that is, does not appear at any ancestor node. By lower-bound results for register automata on data words in Neven et al. [2004], David [2004], and Demri and Lazić [2009], we have that ATRA_1 nonemptiness is not primitive recursive, and that it becomes undecidable (more precisely, r.e.-hard) if any of the following is added: upward moves, leftward moves, or one more register.

Motivated partly by applications to XML streams (cf., e.g., Olteanu et al. [2004]), we consider both finite and countably infinite data trees, where horizontal as well as vertical infinity is allowed. For ATRA_1 with the weak acceptance mechanism, the undecidability result over infinite data words [Demri and Lazić 2009] carries over. However, we show that, for safety ATRA_1, which are closed under intersection and union but not complement, inclusion is decidable and not primitive recursive. When a data tree is rejected by an automaton with the safety acceptance mechanism, there exists an initial segment whose every extension is rejected. We also obtain that nonemptiness of safety ATRA_1 is not elementary. The latter is the most surprising result in the article: it means that the techniques in the proof that nonemptiness over infinite data words of safety one-way alternating automata with 1 register is in ExpSpace cannot be lifted to trees to obtain a 2ExpTime upper bound.

The proofs of decidability involve translating from ATRA_1 to forward nondeterministic tree automata with faulty counters. The counters are faulty in the sense that they are subject to incrementing errors, that is, can spontaneously increase at any time. That makes the transition relations downwards compatible with a well-quasi-ordering (cf. Finkel and Schnoebelen [2001]), which leads to lower complexities of some verification problems than with error-free counters.

We define forward XPath to be the largest downward and rightward fragment in which, whenever two attribute values are compared for equality, one of them must be at the current node. By translating from forward XPath to ATRA_1, we obtain decidability of satisfiability over finite documents and decidability of satisfiability for a safety subfragment, both in the presence of DTDs. In contrast to the decidable fragments of XPath mentioned previously, forward XPath has sibling axes, recursive axes, concatenation, negation, and data comparisons.

2. PRELIMINARIES

After fixing notations for trees and data trees, we define two kinds of forward automata and look at some of their basic properties: alternating automata with 1 register on data trees, and nondeterministic automata with counters with incrementing errors on trees.

2.1. Trees and Data Trees

For technical simplicity, we shall work with binary trees instead of unranked ordered trees. First, as, for example, Björklund and Bojańczyk [2007], we adopt the insignificant generalization of considering unranked ordered forests, in which the roots are regarded as siblings with no parent. Secondly, the following is a standard and trivial one-to-one correspondence between unranked ordered forests t and binary trees $\mathrm{bt}(t)$: the nodes of

bt(t) are the same as the nodes of t, and the children of each node n in bt(t) are the first child and next sibling of n in t. The correspondence works for finite as well as infinite unranked ordered forests. In the latter, there may be infinite (of type ω) branches or siblinghoods or both.

Without loss of generality, each node will either have both children or be a leaf, only nonleaf nodes will be labeled, and the root node will be nonleaf. Formally, a *tree* is a tuple $\langle N, \Sigma, \Lambda \rangle$, where:

—N is a prefix-closed subset of $\{0,1\}^*$ such that $|N| > 1$ and, for each $n \in N$, either $n \cdot 0 \in N$ and $n \cdot 1 \in N$, or $n \cdot 0 \notin N$ and $n \cdot 1 \notin N$;
—Σ is a finite alphabet;
—Λ is a mapping from the nonleaf elements of N to Σ.

A *data tree* is a tree as just described together with a mapping Δ from the nonleaf nodes to a fixed infinite set $\mathcal{D}$. For a data tree τ, let tree(τ) denote the underlying tree.

For a data tree τ and $l > 0$, let the *l-prefix* of τ be the data tree obtained by restricting τ to nodes of length at most l. For each Σ, the set of all data trees with alphabet Σ is a complete metric space with the following notion of distance: for distinct τ and τ', let $d(\tau, \tau') = 1/l$ where l is least such that τ and τ' have distinct l-prefixes.

2.2. Alternating Tree Register Automata

Automata. A run of a forward alternating automaton with 1 register on a data tree will consist of a *configuration* for each tree node. Each configuration will be a finite set of *threads*, which are pairs of an automaton state and a register value, where the latter is a datum from $\mathcal{D}$.

Following Brzozowski and Leiss [1980], transitions will be specified by positive Boolean formulae. For a set of states Q, let $\mathcal{B}^+(Q)$ consist of all formulae given by the following grammar, where $q \in Q$:

$$\varphi \ ::= \ q(0,\downarrow) \mid q(0,\not\downarrow) \mid q(1,\downarrow) \mid q(1,\not\downarrow) \mid \top \mid \bot \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

Given a configuration G at a nonleaf tree node n, for each thread $\langle q, D \rangle$ in G, the automaton transition function provides a formula φ in $\mathcal{B}^+(Q)$, which depends on q, on the letter labeling n, and on whether $D = E$, where E is the datum labeling n. In φ, an atom $r(d, \downarrow)$ requires that thread $\langle r, E \rangle$ be in the configuration for node $n \cdot d$ (i.e., the register value is replaced by the datum at n), and an atom $r(d, \not\downarrow)$ requires the same for thread $\langle r, D \rangle$ (i.e., the register value is not replaced).

Formally, a *forward alternating tree* 1*-register automaton* (shortly, $ATRA_1$) $\mathcal{A}$ is a tuple $\langle \Sigma, Q, q_I, F, \delta \rangle$ such that:

—Σ is a finite alphabet and Q is a finite set of states;
—$q_I \in Q$ is the initial state and $F \subseteq Q$ are the final states;
—$\delta : Q \times \Sigma \times \{tt, ff\} \to \mathcal{B}^+(Q)$ is a transition function.

Runs and Languages. The semantics of the positive Boolean formulae can be given by defining when a quadruple $R_0^{\downarrow}, R_0^{\not\downarrow}, R_1^{\downarrow}, R_1^{\not\downarrow}$ of subsets of Q satisfies a formula φ in $\mathcal{B}^+(Q)$, by structural recursion. The cases for the Boolean atoms and operators are standard, and for the remaining atoms we have:

$$R_0^{\downarrow}, R_0^{\not\downarrow}, R_1^{\downarrow}, R_1^{\not\downarrow} \models r(d,?) \stackrel{\text{def}}{\Leftrightarrow} r \in R_d^?.$$

We can now define the transition relation of $\mathcal{A}$, which is between configurations and pairs of configurations, and relative to a letter and a datum. We write $G \to_a^E H_0, H_1$ iff, for each thread $\langle q, D \rangle \in G$, there exist $R_0^{\downarrow}, R_0^{\not\downarrow}, R_1^{\downarrow}, R_1^{\not\downarrow} \models \delta(q, a, D = E)$ such that, for

both $d \in \{0, 1\}$:

$$\{\langle r, E\rangle \,:\, r \in R_d^{\downarrow}\} \cup \{\langle r, D\rangle \,:\, r \in R_d^{\not\downarrow}\} \subseteq H_d.$$

A run of $\mathcal{A}$ on a data tree $\langle N, \Sigma, \Lambda, \Delta\rangle$ is a mapping $n \mapsto G_n$ from the nodes to configurations such that:

—the initial thread is in the configuration at the root, that is, $\langle q_I, \Delta(\varepsilon)\rangle \in G_\varepsilon$;
—for each nonleaf n, the transition relation is observed, that is, $G_n \to_{\Lambda(n)}^{\Delta(n)} G_{n\cdot 0}, G_{n\cdot 1}$.

We say that the run is:

—*final* iff, for each leaf n, only final states occur in G_n;
—*finite* iff there exists l such that, for each n of length at least l, G_n is empty.

We may regard $\mathcal{A}$ as an automaton on finite data trees, a safety automaton, or a co-safety automaton. We say that:

—$\mathcal{A}$ *accepts* a finite data tree τ iff $\mathcal{A}$ has a final run on τ;
—$\mathcal{A}$ *safety-accepts* a data tree τ iff $\mathcal{A}$ has a final run on τ;
—$\mathcal{A}$ *co-safety-accepts* a data tree τ iff $\mathcal{A}$ has a final finite run on τ.

Observe that, for finite data trees, the three modes of $\mathcal{A}$ coincide.

Let $\mathrm{L}^{fin}(\mathcal{A})$ denote the set of all finite data trees with alphabet Σ that $\mathcal{A}$ accepts, and $\mathrm{L}^{saf}(\mathcal{A})$ (respectively, $\mathrm{L}^{cos}(\mathcal{A})$) denote the set of all data trees with alphabet Σ that $\mathcal{A}$ safety-accepts (respectively, co-safety-accepts).

Remark 2.1. The valid initial and successor configurations in runs were defined in terms of lower bounds on sets. In other words, while running on any data tree, at each node the automaton is free to introduce arbitrary "junk" threads. However, final and finite runs were defined in terms of upper bounds on sets, so junk threads can only make it harder to complete a partial run into an accepting one. This will play an important role in the proof of decidability in Theorem 3.1.

Boolean Operations. Given an ATRA_1 $\mathcal{A}$, let $\overline{\mathcal{A}}$ denote its dual: the automaton obtained by replacing the set of final states with its complement and replacing, in each transition formula $\delta(q, a, p)$, every $\top$ with $\bot$, every $\wedge$ with $\vee$, and vice-versa. Observe that $\overline{\overline{\mathcal{A}}} = \mathcal{A}$. Considering $\mathcal{A}$ (respectively, $\overline{\mathcal{A}}$) as a weak alternating automaton whose every state is of even (respectively, odd) parity, we have by Löding and Thomas [2000, Theorem 1] that $\mathrm{L}^{cos}(\overline{\mathcal{A}})$ is the complement of $\mathrm{L}^{saf}(\mathcal{A})$. Hence, we also have that $\mathrm{L}^{saf}(\overline{\mathcal{A}})$ is the complement of $\mathrm{L}^{cos}(\mathcal{A})$, and that $\mathrm{L}^{fin}(\overline{\mathcal{A}})$ is the complement of $\mathrm{L}^{fin}(\mathcal{A})$.

For each m of *fin*, *saf*, *cos*, given ATRA_1 $\mathcal{A}_1$ and $\mathcal{A}_2$ with alphabet Σ, an automaton whose language in mode m is $\mathrm{L}^{\mathrm{m}}(\mathcal{A}_1) \cap \mathrm{L}^{\mathrm{m}}(\mathcal{A}_2)$ (respectively, $\mathrm{L}^{\mathrm{m}}(\mathcal{A}_1) \cup \mathrm{L}^{\mathrm{m}}(\mathcal{A}_2)$) is constructible easily. It suffices to form a disjoint union of $\mathcal{A}_1$ and $\mathcal{A}_2$, and add a new initial state q_I such that $\delta(q_I, a, tt) = \delta(q_I^1, a, tt) \wedge \delta(q_I^2, a, tt)$ (respectively, $\delta(q_I, a, tt) = \delta(q_I^1, a, tt) \vee \delta(q_I^2, a, tt)$) for each $a \in \Sigma$, where q_I^1 and q_I^2 are the initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$. (Since the initial thread's register value is always the root node's datum, the formulae $\delta(q_I, a, ff)$ are irrelevant.)

We therefore obtaining the following.

PROPOSITION 2.2.

(a) $ATRA_1$ on finite data trees are closed under complement, intersection and union.
(b) Safety $ATRA_1$ and co-safety $ATRA_1$ are dual, and each is closed under intersection and union.

In each case, a required automaton is computable in logarithmic space.

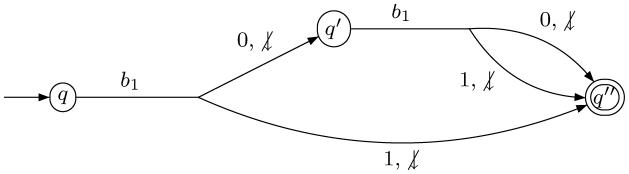

Fig. 1. Defining $\mathcal{B}_1$.

Safety Languages. A set L of data trees with alphabet Σ is called *safety* [Alpern and Schneider 1987] iff it is closed with respect to the metric defined in Section 2.1, that is, for each data tree τ, if for all $l > 0$ there exists $\tau'_l \in L$ such that the l-prefixes of τ and τ'_l are equal, then $\tau \in L$. The complements of safety languages, that is, the open sets of data trees, are called *co-safety*.

PROPOSITION 2.3. *For each ATRA*$_1$ $\mathcal{A}$*, we have that* $\mathrm{L}^{saf}(\mathcal{A})$ *is safety and* $\mathrm{L}^{cos}(\mathcal{A})$ *is co-safety.*

PROOF. By Proposition 2.2(b), it suffices to show that $\mathrm{L}^{saf}(\mathcal{A})$ is safety. Suppose for all $l > 0$ there exists $\tau'_l \in \mathrm{L}^{saf}(\mathcal{A})$ such that the l-prefixes of τ and τ'_l are equal.

For each $l > 0$, let us fix a final run $n \mapsto G'_{l,n}$ of $\mathcal{A}$ on τ'_l. For each $0 \leq k \leq l$, let $\mathcal{G}_{l,k}$ denote the restriction of the run $n \mapsto G'_{l,n}$ to nodes n of length k.

Consider the tree consisting of the empty sequence and all sequences $\mathcal{G}_{l,0} \cdot \mathcal{G}_{l,1} \cdot \cdots \cdot \mathcal{G}_{l,k}$ for $l > 0$ and $0 \leq k \leq l$. Without loss of generality, each register value in each $G'_{l,n}$ labels some node of τ'_l on the path from the root to n, so the tree is finitely branching. By König's Lemma, it has an infinite path $\mathcal{H}_0 \cdot \mathcal{H}_1 \cdot \cdots$. For each $0 \leq k$, $\mathcal{H}_k$ is a mapping from the nodes of τ of length k to configurations of $\mathcal{A}$. It remains to observe that $n \mapsto \mathcal{H}_{|n|}(n)$ is a final run of $\mathcal{A}$ on τ. □

Example 2.4. By recursion on $k \geq 1$, we shall define ATRA_1 $\mathcal{B}_k$ with alphabet $\{b_1, \ldots, b_k, *\}$. As well as being interesting examples of ATRA_1, the $\mathcal{B}_k$ will be used in the nonelementarity part of the proof of Theorem 4.1.

Let $\mathcal{B}_1$ be the automaton depicted in Figure 1. It has three states, where q is initial, and q'' is final. We have $\delta(q, b_1, p) = q'(0, \not\downarrow) \wedge q''(1, \not\downarrow)$ and $\delta(q', b_1, p) = q''(0, \not\downarrow) \wedge q''(1, \not\downarrow)$ for both $p \in \{tt, ff\}$, and the transition function gives $\bot$ in all other cases. (Recalling that the initial thread's register value is the root node's datum, the formula $\delta(q, b_1, ff)$ is in fact irrelevant.) Observe that $\mathcal{B}_1$ safety-accepts exactly data trees that have two nonleaf nodes, the root and its left-hand child, and both are labeled by letter b_1.

For each $k \geq 1$, $\mathcal{B}_{k+1}$ is defined so that it safety-accepts a data tree over $\{b_1, \ldots, b_{k+1}, *\}$ iff:

(i) the root node is labeled by b_{k+1}, its left-hand child is labeled by b_{k+1}, and its right-hand child is a leaf;
(ii) for each node n labeled by b_{k+1}, which is not the root, the left-hand child of n is labeled by $*$ and its both children are labeled by b_{k+1}, and the right-hand subtree at n is safety-accepted by $\mathcal{B}_k$;
(iii) whenever a node n, which is not the root, and a descendant n' of n are labeled by b_{k+1}, we have that their data labels are distinct, and that the datum at n equals the datum at some node which is labeled by b_k and which is in the right-hand subtree at n'.

By Proposition 2.2(b), it suffices to define automata for (i)–(iii) separately. Expressing (i) and (ii) is straightforward, and an automaton for (iii) is depicted in Figure 2. It has four states, where q_0 is initial, and q_1 and q_2 are final. For all letters a and Booleans p, we have $\delta(q_0, a, p) = q_1(0, \not\downarrow)$, so initially the automaton moves to the left-hand child of the root and changes the state to q_1. From q_1, if the current node is labeled by $*$,

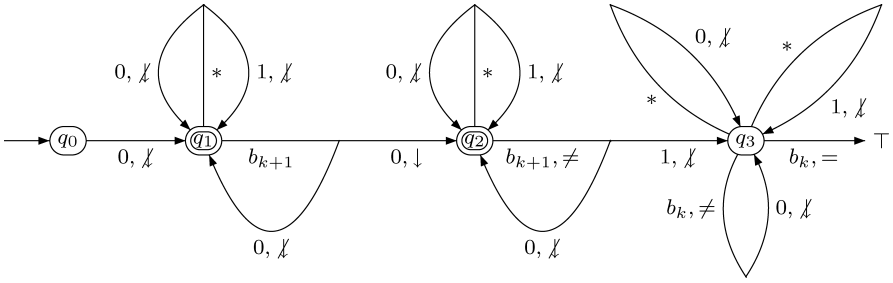

Fig. 2. Defining $\mathcal{B}_{k+1}$.

the automaton moves to both children: $\delta(q_1, *, p) = q_1(0, \not\downarrow) \wedge q_1(1, \not\downarrow)$ for both p. Also from q_1, if the current node n is labeled by b_{k+1}, the automaton both moves to the left-hand child without changing the state, and moves to the left-hand child with storing the datum at n in the register and changing the state to q_2: $\delta(q_1, b_{k+1}, p) = q_1(0, \not\downarrow) \wedge q_2(0, \downarrow)$ for both p. From q_2, the behavior for $*$ is analogous to that from q_1, but if the current node's letter is b_{k+1} and its datum is distinct from the datum in the register, the automaton both moves to the left-hand child without changing the state and moves to the right-hand child with changing the state to q_3: $\delta(q_2, b_{k+1}, \mathit{ff}) = q_2(0, \not\downarrow) \wedge q_3(1, \not\downarrow)$. The remainder of Figure 2 is interpreted similarly, and in cases not depicted, the transition function gives $\bot$. Since the mode of acceptance is safety, the automaton in fact expresses:

(iii′) whenever a node n, which is not the root, and a descendant n' of n are labeled by b_{k+1}, we have that their data labels are distinct, and that either the datum at n equals the datum at some node that is labeled by b_k and which is in the right-hand subtree at n', or that subtree is infinite.

Let $2 \Uparrow 0 = 1$, and $2 \Uparrow k = 2^{2 \Uparrow (k-1)}$ for $k \geq 1$. By induction on $k \geq 1$, the safety language of $\mathcal{B}_k$ has the following two properties. In particular, in the presence of (i) and (ii), we have that (iii) and (iii′) are equivalent.

—for every τ safety-accepted by $\mathcal{B}_k$, every downward sequence that is from the left-hand child of the root and that consists of nodes labeled by b_k is of length at most $2 \Uparrow (k-1)$, so τ is finite and has at most $2 \Uparrow k$ nodes labeled by b_k;
—for some τ safety-accepted by $\mathcal{B}_k$, the nodes labeled by b_k other than the root form a full binary tree of height $2 \Uparrow (k-1)$ (after removing the nodes labeled by $*$), so there are $2 \Uparrow k$ nodes labeled by b_k, and moreover the data at those nodes are mutually distinct.

Finally, we observe that for computing $\mathcal{B}_k$, space logarithmic in k suffices.

2.3. Faulty Tree Counter Automata

In Section 3, we shall establish decidability of nonemptiness of forward alternating tree 1-register automata over finite data trees, by translating them to automata that have natural-valued counters with increments, decrements and zero-tests. The translation will eliminate conjunctive branchings, by having configurations of the former automata (which are finite sets of threads) correspond to pairs of states and counter valuations, so the latter automata will be only nondeterministic. Also, data will be abstracted in the translation, so the counter automata will run on finite trees (without data).

The feature that will make nonemptiness of the counter automata decidable (on finite trees) is that they will be faulty, in the sense that one or more counters can erroneously increase at any time. The key insight is that such faults do not affect the

translation's preservation of nonemptiness: they in fact correspond to introductions of "junk" threads in runs of ATRA_1 (cf. Remark 2.1).

For clarity of the correspondence between the finitary languages of ATRA_1 and the languages of their translations, the counter automata will have ε-transitions.

We now define the counter automata, and show their nonemptiness decidable.

Automata. An *incrementing tree counter automaton* (shortly, *ITCA*) $\mathcal{C}$, which is forward and with ε-transitions, is a tuple $\langle \Sigma, Q, q_I, F, k, \delta \rangle$ such that:

—Σ is a finite alphabet and Q is a finite set of states;
—$q_I \in Q$ is the initial state and $F \subseteq Q$ are the final states;
—$k \in \mathbb{N}$ is the number of counters;
—$\delta \subseteq (Q \times \Sigma \times L \times Q \times Q) \cup (Q \times \{\varepsilon\} \times L \times Q)$ is a transition relation, where $L = \{\mathtt{inc}, \mathtt{dec}, \mathtt{ifz}\} \times \{1, \ldots, k\}$ is the instruction set.

Runs and Languages. A counter valuation is a mapping from $\{1, \ldots, k\}$ to $\mathbb{N}$. For counter valuations v and v', we write:

$$
\begin{array}{rcl}
v \leq v' & \text{iff} & v(c) \leq v'(c) \text{ for all } c \\
v \xrightarrow{\langle \mathtt{inc}, c \rangle}_{\surd} v' & \text{iff} & v' = v[c \mapsto v(c) + 1] \\
v \xrightarrow{\langle \mathtt{dec}, c \rangle}_{\surd} v' & \text{iff} & v' = v[c \mapsto v(c) - 1] \\
v \xrightarrow{\langle \mathtt{ifz}, c \rangle}_{\surd} v' & \text{iff} & v(c) = 0 \text{ and } v' = v \\
v \xrightarrow{l} v' & \text{iff} & v \leq v_{\surd} \xrightarrow{l}_{\surd} v'_{\surd} \leq v' \text{ for some } v_{\surd}, v'_{\surd}
\end{array}
$$

A *configuration* of $\mathcal{C}$ is a pair $\langle q, v \rangle$, where q is a state and v is a counter valuation.

To define runs, we first specify that a *block* is a nonempty finite sequence of configurations obtainable by performing ε-transitions, that is, for every two adjacent configurations $\langle q_i, v_i \rangle$ and $\langle q_{i+1}, v_{i+1} \rangle$ in a block, there exists l with $\langle q_i, \varepsilon, l, q_{i+1} \rangle \in \delta$ and $v_i \xrightarrow{l} v_{i+1}$.

Now, a run of $\mathcal{C}$ on a finite tree $\langle N, \Sigma, \Lambda \rangle$ is a mapping $n \mapsto B_n$ from the nodes to blocks such that:

—$\langle q_I, \mathbf{0} \rangle$ is the first configuration in B_ε;
—for each nonleaf n, there exists l with $\langle q, \Lambda(n), l, r_0, r_1 \rangle \in \delta$, $v \xrightarrow{l} w_0$ and $v \xrightarrow{l} w_1$, where $\langle q, v \rangle$ is the last configuration in B_n, and $\langle r_0, w_0 \rangle$ and $\langle r_1, w_1 \rangle$ are the first configurations in $B_{n \cdot 0}$ and $B_{n \cdot 1}$ (respectively).

We regard such a run accepting iff, for each leaf n, the state of the last configuration in B_n is final. The language $\mathrm{L}(\mathcal{C})$ is the set of all finite trees with alphabet Σ on which $\mathcal{C}$ has an accepting run.

Decidability of Nonemptiness. We remark that, since nonemptiness of incrementing counter automata over words is not primitive recursive [Demri and Lazić 2009, Theorem 2.9(b)], the same is true of nonemptiness of ITCA.

THEOREM 2.5. *Nonemptiness of ITCA is decidable.*

PROOF. Consider an ITCA $\mathcal{C} = \langle \Sigma, Q, q_I, F, k, \delta \rangle$.

For counter valuations v and v', and an instruction l, we say that v under l yields v' *lazily* and write $v \xrightarrow{l}_{\flat} v'$ iff either $v \xrightarrow{l}_{\surd} v'$ (that is, there are no incrementing errors), or l is of the form $\langle \mathtt{dec}, c \rangle$, $v(c) = 0$ and $v' = v$ (that is, 0 is erroneously decremented to 0). Observe that:

(*) Whenever $v \leq w$ and $w \xrightarrow{l} w'$, there exists v' such that $v \xrightarrow{l}_{\flat} v'$ and $v' \leq w'$.

To reduce the nonemptiness problem for $\mathcal{C}$ to a reachability problem, let a *level* of $\mathcal{C}$ be a finite set of configurations. For levels $\mathcal{G}$ and $\mathcal{G}'$ of $\mathcal{C}$, let us write $\mathcal{G} \to \mathcal{G}'$ iff $\mathcal{G}'$ can be obtained from $\mathcal{G}$ as follows.

—Each $\langle q, v\rangle \in \mathcal{G}$ with $q \notin F$ is replaced either by the two configurations that some firable transition $\langle q, a, l, r_0, r_1\rangle$ yields lazily, or by the one configuration that some firable transition $\langle q, \varepsilon, l, r\rangle$ yields lazily.
—Each $\langle q, v\rangle \in \mathcal{G}$ with $q \in F$ is removed.

Performing transitions of $\mathcal{C}$ lazily ensures that, for every level $\mathcal{G}$, the set $\{\mathcal{G}' : \mathcal{G} \to \mathcal{G}'\}$ of all its successors is finite. The latter set is also computable. By the definition of accepting runs and (*), we have that $\mathcal{C}$ is nonempty iff the empty level is reachable from the initial level $\{\langle q_I, \mathbf{0}\rangle\}$.

For configurations $\langle q, v\rangle$ and $\langle r, w\rangle$, let $\langle q, v\rangle \leq \langle r, w\rangle$ iff $q = r$ and $v \leq w$. Now, let $\preceq$ be the quasi-ordering obtained by lifting $\leq$ to levels: $\mathcal{G} \preceq \mathcal{H}$ iff, for each $\langle q, v\rangle \in \mathcal{G}$, there exists $\langle r, w\rangle \in \mathcal{H}$ such that $\langle q, v\rangle \leq \langle r, w\rangle$. By Higman's Lemma [Higman 1952], $\preceq$ is a well-quasi-ordering, that is, for every infinite sequence $\mathcal{G}_0, \mathcal{G}_1, \ldots$, there exist $i < j$ such that $\mathcal{G}_i \preceq \mathcal{G}_j$. Observe that, in the terminology of Finkel and Schnoebelen [2001], $\preceq$ is strongly downward-compatible with $\to$: whenever $\mathcal{G} \preceq \mathcal{H}$ and $\mathcal{H} \to \mathcal{H}'$, there exists $\mathcal{G}'$ such that $\mathcal{G} \to \mathcal{G}'$ and $\mathcal{G}' \preceq \mathcal{H}'$. Also, $\preceq$ is decidable.

Since $\mathcal{G} \preceq \emptyset$ iff $\mathcal{G} = \emptyset$, we have reduced nonemptiness of $\mathcal{C}$ to the subcovering problem for downward well-structured transition systems with reflexive (which is weaker than strong) compatibility, computable successor sets and decidable ordering. The latter is decidable by Finkel and Schnoebelen [2001, Theorem 5.5]. □

3. DECIDABILITY OVER FINITE DATA TREES

THEOREM 3.1. *Nonemptiness of ATRA$_1$ over finite data trees is decidable and not primitive recursive.*

PROOF. By considering data words as data trees (e.g., by using only left-hand children starting from the root), the lower bound follows from nonprimitive recursiveness of nonemptiness of one-way co-nondeterministic (that is, with only conjunctive branching) automata with 1 register over finite data words [Demri and Lazić 2009, Theorem 5.2].

We shall establish decidability by reducing to nonemptiness of ITCA, which is decidable by Theorem 2.5. More specifically, by extending to trees the translation in the proof of Demri and Lazić [2009, Theorem 4.4], which is from one-way alternating automata with 1 register on finite data words to incrementing counter automata on finite words, we shall show that, for each ATRA$_1$ $\mathcal{A}$, an ITCA $\mathcal{C}_\mathcal{A}$ with the same alphabet and such that $\mathrm{L}(\mathcal{C}_\mathcal{A}) = \{\mathrm{tree}(\tau) : \tau \in \mathrm{L}^{fin}(\mathcal{A})\}$, is computable (in polynomial space).

Let $\mathcal{A} = \langle \Sigma, Q, q_I, F, \delta\rangle$. For a configuration G of $\mathcal{A}$ and a datum D, let the *bundle* of D in G be the set of all states that are paired with D, that is, $\{q : \langle q, D\rangle \in G\}$. The computation of $\mathcal{C}_\mathcal{A}$ with the properties above is based on the following abstraction of configurations of $\mathcal{A}$ by mappings from $\mathcal{P}(Q) \setminus \{\emptyset\}$ to $\mathbb{N}$. The abstract configuration $\overline{G}$ counts, for each nonempty $S \subseteq Q$, the number of data whose bundles equal S:

$$\overline{G}(S) = |\{D : \{q : \langle q, D\rangle \in G\} = S\}|$$

Thus, two configurations have the same abstraction iff they are equal up to a bijective renaming of data. For $1 \leq i \leq \overline{G}(S)$ and $q \in S$, we shall call pairs $\langle S, i\rangle$ *abstract data* and triples $\langle q, S, i\rangle$ *abstract threads*.

For abstract configurations v, w_0 and w_1, letters a, and sets of states $Q_=$ with either $v(Q_=) > 0$ or $Q_= = \emptyset$, we shall define transitions $v \to_a^{Q_=} w_0, w_1$, and show that they are bisimilar to transitions $G \to_a^E H_0, H_1$ such that $v = \overline{G}$, $w_0 = \overline{H_0}$, $w_1 = \overline{H_1}$ and $Q_= = \{q : \langle q, E\rangle \in G\}$. The sets $Q_=$ can hence be thought of as abstractions of the data

E. The abstract transitions will then give us a notion of abstract run of $\mathcal{A}$ on a finite tree (without data), where the sets $Q_=$ are guessed at every step. By the bisimilarity, we shall have that:

(I) $\mathcal{A}$ has an accepting abstract run on a finite tree t with alphabet Σ iff it has an accepting run on some data tree τ such that $t = \mathrm{tree}(\tau)$.

In other words, we shall have reduced the question of whether $\mathrm{L}^{fin}(\mathcal{A})$ is nonempty, that is, whether there exists a finite tree with alphabet Σ, a data labeling of its nonleaf nodes, and an accepting run of $\mathcal{A}$ on the resulting data tree, to whether there exists a finite tree and an accepting abstract run of $\mathcal{A}$ on it. It will then remain to show how to compute (in polynomial space) an ITCA $\mathcal{C}_\mathcal{A}$ which guesses and checks accepting abstract runs of $\mathcal{A}$, so that:

(II) $\mathcal{C}_\mathcal{A}$ has an accepting run on a finite tree t with alphabet Σ iff $\mathcal{A}$ has an accepting abstract run on t.

To begin delivering our promises, we now define transitions from abstract configurations v for letters a and sets of states $Q_=$ with either $v(Q_=) > 0$ or $Q_= = \emptyset$ to abstract configurations w_0 and w_1, essentially by reformulating the definition of concrete transitions (cf. Section 2.2) in terms of abstract threads. For each abstract datum $\langle S, i\rangle$ of v and both $d \in \{0, 1\}$, the abstract threads whose abstract datum is $\langle S, i\rangle$ will contribute two sets of states to such a transition: $R'(S, i)^{\downarrow}_d$, for which the automaton's register is updated, and $R'(S, i)^{\not\downarrow}_d$, for which the automaton's register is not updated. If $Q_=$ is nonempty, we take $\langle Q_=, 1\rangle$ to represent the datum abstracted by $Q^=$, that is, with which the register is updated, so states in the union of the set $R'(Q_=, 1)^{\not\downarrow}_d$ and all the sets $R'(S, i)^{\downarrow}_d$ will be associated to the same abstract datum of w_d. Formally, let $v \to^{Q_=}_a w_0, w_1$ mean that, for each abstract datum $\langle S, i\rangle$ of v, there exist sets of states $R'(S, i)^{\downarrow}_0, R'(S, i)^{\not\downarrow}_0, R'(S, i)^{\downarrow}_1, R'(S, i)^{\not\downarrow}_1$ such that:

(i) for each abstract thread $\langle q, S, i\rangle$ of v, there exist

$$R^{\downarrow}_0, R^{\not\downarrow}_0, R^{\downarrow}_1, R^{\not\downarrow}_1 \models \delta(q, a, \langle S, i\rangle = \langle Q_=, 1\rangle)$$

which satisfy $R^{?}_d \subseteq R'(S, i)^{?}_d$ for both $d \in \{0, 1\}$ and $? \in \{\downarrow, \not\downarrow\}$;

(ii) for both $d \in \{0, 1\}$ and each nonempty $S' \subseteq Q$, we have

$$|\{\langle S, i\rangle \,:\, \langle S, i\rangle \neq \langle Q_=, 1\rangle \wedge R'(S, i)^{\not\downarrow}_d = S'\}| + \left\{ \begin{array}{ll} 1, & \text{if } R^{=}_d = S' \\ 0, & \text{otherwise} \end{array} \right\} \leq w_d(S')$$

for some $R^{=}_d \supseteq R'(Q_=, 1)^{\not\downarrow}_d \cup \bigcup_{1 \leq i \leq v(S)} R'(S, i)^{\downarrow}_d$.

It is straightforward to check the following two-part correspondence between the abstract transitions just defined and concrete transitions.

(IIIa) Whenever $G \to^{E}_a H_0, H_1$, we have $v \to^{Q_=}_a w_0, w_1$, where $v = \overline{G}$, $w_0 = \overline{H_0}$, $w_1 = \overline{H_1}$ and $Q_= = \{q : \langle q, E\rangle \in G\}$.

(IIIb) Whenever $\overline{G} = v$ and $v \to^{Q_=}_a w_0, w_1$, there exist E, H_0 and H_1 such that $G \to^{E}_a H_0, H_1$, $w_0 = \overline{H_0}$, $w_1 = \overline{H_1}$ and $Q_= = \{q : \langle q, E\rangle \in G\}$.

Let α be a bijection between the abstract data of v and the data that occur in G, which is bundle preserving (that is, whenever $\alpha\langle S, i\rangle = D$, we have that S is the bundle of D in G), and if $Q_=$ is nonempty then $\alpha\langle Q_=, 1\rangle = E$.

—To show (IIIa), for each abstract datum $\langle S, i\rangle$ of v and both $d \in \{0, 1\}$, take $R'(S, i)^{\downarrow}_{d}$ and $R^{=}_{d}$ to be the bundle of E in H_d, and take $R'(S, i)^{\not\downarrow}_{d}$ to be the bundle of $\alpha\langle S, i\rangle$ in H_d.
—For (IIIb), if $Q_{=}$ is empty then take E to be an arbitrary datum that does not occur in G, pick the same quadruples for the threads in G as for the corresponding (via α) abstract threads of v, and for both $d \in \{0, 1\}$, obtain H_d from w_d by replacing each set of abstract data $\langle S', 1\rangle, \ldots, \langle S', w_d(S')\rangle$ with: the data $\alpha\langle S, i\rangle$ such that $\langle S, i\rangle \neq \langle Q_{=}, 1\rangle$ and $R'(S, i)^{\not\downarrow}_{d} = S'$, the datum E if $R^{=}_{d} = S'$, and fresh further data if the inequality in (ii) is strict.

Composing abstract transitions gives us abstract runs of $\mathcal{A}$. Such a run on a finite tree $\langle N, \Sigma, \Lambda\rangle$ is a mapping $n \mapsto v_n$ from the nodes to abstract configurations such that, for each nonleaf n, there exists $Q_{=}$ with $v_n \rightarrow^{Q_{=}}_{\Lambda(n)} v_{n\cdot 0}, v_{n\cdot 1}$, and if n is the root then $q_I \in Q_{=}$. Defining the run to be accepting iff $v_n(S) = 0$ for all leaves n and all $S \not\subseteq F$, we have (I) above by (IIIa) and (IIIb).

We are now ready to define $\mathcal{C}_{\mathcal{A}}$, as an ITCA that performs the steps (1)–(9) that follow. States of $\mathcal{C}_{\mathcal{A}}$ are used for control and for storing a, $Q_{=}$, *root* (initially *tt*), S, $R'^{\downarrow}_{0}$, $R'^{\not\downarrow}_{0}$, $R'^{\downarrow}_{1}$, $R'^{\not\downarrow}_{1}$, q, $R^{\downarrow}_{0}$, $R^{\not\downarrow}_{0}$, $R^{\downarrow}_{1}$, $R^{\not\downarrow}_{1}$, d, ? and $R^{=}_{d}$. There are $2^{|Q|} - 1$ counters in the array c, and $2^{|Q|^4}$ counters in the array c'. Steps are implemented by ε-transitions, except for the a-transition in (4). The choices are nondeterministic. If a choice in (3.2) is impossible, or a check in (2), (3.2) or (5) fails, then $\mathcal{C}_{\mathcal{A}}$ blocks.

Steps (1)–(9) guess and check an accepting abstract run of $\mathcal{A}$ on a finite tree. The counter array c is used to store abstract configurations, and the counter array c' is auxiliary. The initial condition in the definition of abstract runs is checked in (2), the final condition in (8), and steps (3)–(7) are essentially a reformulation of the definition of abstract transitions. This particular reformulation is tailored for a development in the proof of Theorem 4.1, and is based on observing that the quadruples of sets $R'(S, i)^{\downarrow}_{0}, R'(S, i)^{\not\downarrow}_{0}, R'(S, i)^{\downarrow}_{1}, R'(S, i)^{\not\downarrow}_{1}$ for abstract data $\langle S, i\rangle \neq \langle Q_{=}, 1\rangle$ do not need to be stored simultaneously, that is, that it suffices to store numbers of such identical quadruples, which is done using the counter array c'.

(1) Choose $a \in \Sigma$, and $Q_{=}$ with either $c[Q_{=}] > 0$ or $Q_{=} = \emptyset$.
(2) If $root = tt$, then check that $q_I \in Q_{=}$ and set $root := ff$.
(3) For each nonempty $S \subseteq Q$, while $c[S] > 0$ do:
  (3.1) choose $R'^{\downarrow}_{0}, R'^{\not\downarrow}_{0}, R'^{\downarrow}_{1}, R'^{\not\downarrow}_{1} \subseteq Q$;
  (3.2) for each $q \in S$, choose $R^{\downarrow}_{0}, R^{\not\downarrow}_{0}, R^{\downarrow}_{1}, R^{\not\downarrow}_{1} \models \delta(q, a, \langle S, c[S]\rangle = \langle Q_{=}, 1\rangle)$, and check that $R^{?}_{d} \subseteq R'^{?}_{d}$ for both $d \in \{0, 1\}$ and $? \in \{\downarrow, \not\downarrow\}$;
  (3.3) decrement $c[S]$, and if $\langle S, c[S]\rangle = \langle Q_{=}, 0\rangle$, then choose $R^{=}_{d} \supseteq R'^{\downarrow}_{d} \cup R'^{\not\downarrow}_{d}$ for both $d \in \{0, 1\}$, else increment $c'[R'^{\downarrow}_{0}, R'^{\not\downarrow}_{0}, R'^{\downarrow}_{1}, R'^{\not\downarrow}_{1}]$.
(4) Perform an a-transition, forking with $d := 0$ and $d := 1$.
(5) Check that $R^{=}_{d} \supseteq \bigcup\{R'^{\downarrow}_{d} : c'[R'^{\downarrow}_{0}, R'^{\not\downarrow}_{0}, R'^{\downarrow}_{1}, R'^{\not\downarrow}_{1}] > 0\}$, and increment $c[R^{=}_{d}]$.
(6) Transfer each $c'[R'^{\downarrow}_{0}, R'^{\not\downarrow}_{0}, R'^{\downarrow}_{1}, R'^{\not\downarrow}_{1}]$ with nonempty $R'^{\not\downarrow}_{d}$ to $c[R'^{\not\downarrow}_{d}]$.
(7) Reset (that is, decrement until 0) each $c'[R'^{\downarrow}_{0}, R'^{\not\downarrow}_{0}, R'^{\downarrow}_{1}, R'^{\not\downarrow}_{1}]$ with empty $R'^{\not\downarrow}_{d}$.
(8) If $c[S] = 0$ whenever $S \not\subseteq F$, then pass through a final state.
(9) Repeat from (1).

Since $\mathcal{C}_{\mathcal{A}}$ is an ITCA, its runs may contain arbitrary errors that increase one or more counters. Nevertheless, between executions of steps (3)–(7) by $\mathcal{C}_{\mathcal{A}}$ and abstract transitions of $\mathcal{A}$, we have the following two-part correspondence. It shows that the

possibly erroneous executions of (3)–(7) match the abstract transitions with the slack allowed by their definition, which in turn match the concrete transitions with their possible introductions of junk threads (cf. (IIIa), (IIIb) and Remark 2.1).

(IVa) Whenever $v \rightarrow_a^{Q_=} w_0, w_1$, we have that $\mathcal{C}_\mathcal{A}$ can perform steps (3)–(7) beginning with any configuration such that each $c[S]$ has value $v(S)$ and each $c'[R'^{\downarrow}_0, R'^{\not\downarrow}_0, R'^{\downarrow}_1, R'^{\not\downarrow}_1]$ has value 0, so that for both forks $d \in \{0, 1\}$ in (4), the ending configuration is such that each $c[S]$ has value $w_d(S)$ and each $c'[R'^{\downarrow}_0, R'^{\not\downarrow}_0, R'^{\downarrow}_1, R'^{\not\downarrow}_1]$ has value 0.

(IVb) Whenever $\mathcal{C}_\mathcal{A}$ can perform steps (3)–(7) beginning with a configuration such that a and $Q_=$ are as in (1) and each $c[S]$ has value $v(S)$, so that for both forks $d \in \{0, 1\}$ in (4), the ending configuration is such that each $c[S]$ has value $w_d(S)$, we have $v \rightarrow_a^{Q_=} w_0, w_1$.

—In proving (IVa), we can choose where incrementing errors occur. For each iteration of (3.1)–(3.3), let the quadruple chosen in (3.1) be

$$R'(S, c[S])^{\downarrow}_0, R'(S, c[S])^{\not\downarrow}_0, R'(S, c[S])^{\downarrow}_1, R'(S, c[S])^{\not\downarrow}_1$$

so that (3.2) can succeed by (i) in the definition of abstract transitions. It remains to match by incrementing errors, say at the end of (7), any differences between the two sides of the inequalities in (ii).

—To obtain (IVb), let $R'(S, i)^{\downarrow}_0, R'(S, i)^{\not\downarrow}_0, R'(S, i)^{\downarrow}_1, R'(S, i)^{\not\downarrow}_1$ for each abstract datum $\langle S, i\rangle$ of v be the quadruple chosen in the last performance of (3.1) with $i = c[S]$ (due to incrementing errors, there may be more than one). Step (3.2) ensures that (i) is satisfied. Since at the end of (3), each $c'[R'^{\downarrow}_0, R'^{\not\downarrow}_0, R'^{\downarrow}_1, R'^{\not\downarrow}_1]$ has value at least

$$|\{\langle S, i\rangle \,:\, \langle S, i\rangle \neq \langle Q_=, 1\rangle \wedge \forall d, ?(R'(S, i)^?_d = R'^?_d)\}|$$

steps (5) and (6) ensure that (ii) is satisfied.

Now, we have (II) from before. The 'if' direction follows by (IVa), and the 'only if' direction by (IVb) once we observe that, without loss of generality, we can consider only runs of $\mathcal{C}_\mathcal{A}$ that do not contain incrementing errors on the array c outside of steps (3)–(7) except before the first performance of (1).

To conclude that polynomial space suffices for computing $\mathcal{C}_\mathcal{A}$, we observe that each of its state variables is either from a fixed finite set, or an element of Σ, or an element or subset of Q, and that deciding satisfaction of transition formulae $\delta(q, a, \langle S, c[S]\rangle = \langle Q_=, 1\rangle)$ in step (3.2) amounts to evaluating Boolean formulae. □

We remark that, in the opposite direction to the translation in the proof of Theorem 3.1, by extending the proof of Demri and Lazić [2009, Theorem 5.2] to trees, for each ITCA $\mathcal{C}$, an ATRA_1 $\mathcal{A}_\mathcal{C}$ is computable in logarithmic space such that $\mathrm{L}^{fin}(\mathcal{A}_\mathcal{C})$ consists of encodings of accepting runs of $\mathcal{C}$. Moreover, similarly as on words, the two translations can be extended to infinite trees, where ATRA_1 are equipped with weak acceptance and ITCA with Büchi acceptance. Instead of decidable and not primitive recursive as on finite trees, nonemptiness for those two classes of automata can then be shown co-r.e.-complete.

4. SAFETY AUTOMATA

We now show decidability of nonemptiness of forward alternating tree 1-register automata with safety acceptance over finite or infinite data trees. More precisely, since the class of safety ATRA_1 is not closed under complement, but is closed under intersection and union (cf. Proposition 2.2(b)), we show decidability of the inclusion problem,

which implies decidability of nonemptiness of Boolean combinations of safety ATRA_1. However, already for the subproblems of nonemptiness and nonuniversality, we obtain nonelementary and nonprimitive recursive lower bounds (respectively).

THEOREM 4.1. *For safety $ATRA_1$, inclusion is decidable, nonemptiness is not elementary, and nonuniversality is not primitive recursive.*

PROOF. Showing that the inclusion problem is decidable will involve extending:

—the proof of Proposition 2.2 to obtain an intersection of a safety and a co-safety ATRA_1, which can be seen as a weak parity ATRA_1 with 2 priorities;
—the proof of Theorem 3.1 to obtain an ITCA with a more powerful set of instructions and no cycles of ε-transitions, which can also be seen as having weak parity acceptance with 2 priorities;
—the proof of Theorem 2.5 to obtain decidability of nonemptiness of such ITCA.

To maintain focus, we shall avoid introducing the extended notions in general, but concentrate on what is necessary for this part of the proof.

Suppose $\mathcal{A}_1 = \langle \Sigma, Q_1, q_I^1, F_1, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, Q_2, q_I^2, F_2, \delta_2 \rangle$ are ATRA_1, where we need to determine whether $\mathrm{L}^{saf}(\mathcal{A}_1)$ is a subset of $\mathrm{L}^{saf}(\mathcal{A}_2)$. By the proof of Proposition 2.2(b), that amounts to emptiness of the intersection of $\mathrm{L}^{saf}(\mathcal{A}_1)$ and $\mathrm{L}^{cos}(\overline{\mathcal{A}_2})$, where $\overline{\mathcal{A}_2} = \langle \Sigma, Q_2, q_I^2, \overline{F_2}, \overline{\delta_2} \rangle$ is the dual automaton to $\mathcal{A}_2$. Assuming that Q_1 and Q_2 are disjoint, and do not contain $q_I^\cap$, let

$$\mathcal{A}_\cap = \langle \Sigma, \{q_I^\cap\} \cup Q_1 \cup Q_2, q_I^\cap, F_1 \cup F_2, \delta_\cap \rangle$$

be the automaton for the intersection of $\mathcal{A}_1$ and $\overline{\mathcal{A}_2}$:

$$\delta_\cap(q, a, p) = \begin{cases} \delta(q_I^1, a, p) \wedge \delta(q_I^2, a, p), & \text{if } q = q_I^\cap \\ \delta_1(q, a, p), & \text{if } q \in Q_1 \\ \overline{\delta_2}(q, a, p), & \text{if } q \in Q_2 \end{cases}$$

We then have:

(*) A data tree τ with alphabet Σ is safety-accepted by $\mathcal{A}_1$ and co-safety-accepted by $\overline{\mathcal{A}_2}$ iff $\mathcal{A}_\cap$ has a run on τ which is final and Q_2-finite, that is, there exists l such that the configuration at each node of length at least l contains no threads with states from Q_2.

Before proceeding, let incrementing tree counter automata with nondeterministic transfers (ITCANT) be defined as ITCA (cf. Section 2.3), except that $\langle \mathtt{ifz}, c \rangle$ instructions are replaced by $\langle \mathtt{transf}, c, C \rangle$ for counters c and sets of counters C. Such an instruction is equivalent to a loop that executes while c is nonzero, and in each iteration, decrements c and increments some counter in C. However, in presence of incrementing errors, the loop may not terminate, whereas $\langle \mathtt{transf}, c, C \rangle$ instructions are considered atomic. The effect of $\langle \mathtt{transf}, c, C \rangle$ is therefore to transfer the value of c to the counters in C, among which it is split nondeterministically. In particular, $\langle \mathtt{ifz}, c \rangle$ instructions can be reintroduced as $\langle \mathtt{transf}, c, \emptyset \rangle$.

Now, steps (1)–(9) in the proof of Theorem 3.1 can be implemented by an ITCANT which uses nondeterministic transfers instead of the loops in (3), (6) and (7), and whose transition relation therefore contains no cycles of ε-transitions. More specifically, each reset in (7) can be implemented as a transfer to a new auxiliary counter c'', (6) already consists of transfers to single counters, and (3) can be replaced by the following two steps.

(3a) If $Q_= \neq \emptyset$, then decrement $c[Q_=]$ and choose $R_0^=, R_1^= \subseteq Q$ such that, for each $q \in Q_=$, there exist $R_0^\downarrow, R_0^\not\downarrow, R_1^\downarrow, R_1^\not\downarrow \models \delta(q, a, tt)$ with $R_d^= \supseteq R_d^\downarrow \cup R_d^\not\downarrow$ for both $d \in \{0, 1\}$.

(3b) Transfer each $c[S]$ nondeterministically to the set of all $c'[R'^\downarrow_0, R'^\not\downarrow_0, R'^\downarrow_1, R'^\not\downarrow_1]$ such that, for each $q \in S$, there exist $R_0^\downarrow, R_0^\not\downarrow, R_1^\downarrow, R_1^\not\downarrow \models \delta(q, a, ff)$ with $R_d^? \subseteq R'^?_d$ for both $d \in \{0, 1\}$ and $? \in \{\downarrow, \not\downarrow\}$.

Let $\mathcal{C}_\cap$ be such an ITCANT for $\mathcal{A}_\cap$, which in addition performs the following step between (7) and (8), where *prop* is a state variable, initially ff:

($7\frac{1}{2}$) If $c[S] = 0$ whenever $S \cap Q_2 \neq \emptyset$, then set $prop := tt$.

As in the proof of Theorem 3.1, we have that $\mathcal{C}_\cap$ is computable from $\mathcal{A}_\cap$, and therefore from $\mathcal{A}_1$ and $\mathcal{A}_2$, in polynomial space. Also, $\mathcal{A}_\cap$ satisfies (IIIa) and (IIIb), and $\mathcal{A}_\cap$ and $\mathcal{C}_\cap$ satisfy (IVa) and (IVb). Recalling that $\mathcal{C}_\cap$ contains no cycles of ε-transitions, we infer the following from (*) above, where the notion of transitions between levels of $\mathcal{C}_\cap$ is as in the proof of Theorem 2.5, and P denotes the set of all states of $\mathcal{C}_\cap$ in which *prop* has value tt:

(**) $\mathrm{L}^{saf}(\mathcal{A}_1)$ is a subset of $\mathrm{L}^{saf}(\mathcal{A}_2)$ iff there does not exist an infinite sequence of transitions $\mathcal{G}_0 \to \mathcal{G}_1 \to \cdots$ which is from the initial level of $\mathcal{C}_\cap$ and such that some $\mathcal{G}_i$ contains only states from P.

To conclude decidability of inclusion, we show that, given an ITCANT $\mathcal{C}_\cap$ and a set P of its states, existence of an infinite sequence of transitions as in (**) is decidable. For a set $\mathbb{G}$ of levels of $\mathcal{C}_\cap$, we write $\uparrow\mathbb{G}$ to denote its upward closure with respect to $\preceq$: the set of all $\mathcal{H}$ for which there exists $\mathcal{G} \in \mathbb{G}$ with $\mathcal{G} \preceq \mathcal{H}$. We say that $\mathbb{G}$ is upwards closed iff $\mathbb{G} = \uparrow\mathbb{G}$, and we say that $\mathbb{H}$ is a basis for $\mathbb{G}$ iff $\mathbb{G} = \uparrow\mathbb{H}$. As in the proof of Theorem 2.5, we have that successor sets with respect to $\to$ are computable, $\preceq$ is a well-quasi-ordering, $\preceq$ is strongly (in particular, reflexively) downward-compatible with $\to$, and $\preceq$ is decidable. Hence, by Finkel and Schnoebelen [2001, Proposition 5.4], a finite basis $\mathbb{G}_R$ of the upward closure of the set of all levels reachable from the initial level is computable. By the strong downward compatibility, the set of all levels from which there exists an infinite sequence of transitions is downwards closed, so its complement is upwards closed. We claim that a finite basis $\mathbb{G}_T$ of the latter set is computable. With that assumption, since a finite basis $\mathbb{G}_N$ of the set of all levels that contain some state not from P is certainly computable, we are done because there does not exist an infinite sequence of transitions as in (**) iff $\uparrow\mathbb{G}_R$ is a subset of the union of $\uparrow\mathbb{G}_T$ and $\uparrow\mathbb{G}_N$.

It remains to establish the claim. For a finite set $\mathbb{G}'$ of levels of $\mathcal{C}_\cap$, let

$$K(\mathbb{G}') = 1 + \max_{\mathcal{G}' \in \mathbb{G}'} \max_{\langle q, v\rangle \in \mathcal{G}'} \sum_{c \in \{1,\ldots,k\}} v(c)$$

where k is the number of counters of $\mathcal{C}_\cap$. Let also $\mathrm{Pred}_\forall(\mathbb{G}')$ be the upwards-closed set consisting of all $\mathcal{G}$ such that, whenever $\mathcal{G} \to \mathcal{G}'$, we have $\mathcal{G}' \in \uparrow\mathbb{G}'$. Observe that, whenever $\mathcal{G} \in \mathrm{Pred}_\forall(\mathbb{G}')$, there exists $\mathcal{G}_\dagger \in \mathrm{Pred}_\forall(\mathbb{G}')$ such that $\mathcal{G}_\dagger \preceq \mathcal{G}$ and, for each $\langle q, v\rangle \in \mathcal{G}_\dagger$ and $c \in \{1, \ldots, k\}$, $v(c) \leq K(\mathbb{G}')$. Hence, a finite basis of $\mathrm{Pred}_\forall(\mathbb{G}')$ is computable, so the following is an effective procedure:

(i) Begin with $\mathbb{G}_T := \emptyset$.
(ii) Let $\mathbb{H}$ be a finite basis of $\mathrm{Pred}_\forall(\mathbb{G}_T)$.
(iii) If $\mathbb{H} \not\subseteq \uparrow\mathbb{G}_T$, then set $\mathbb{G}_T := \mathbb{G}_T \cup \mathbb{H}$ and repeat from (ii), else terminate.

$c' := m$; $\texttt{inc}(\overline{c_i})$;
while $c' > 0$
{ $\texttt{dec}(c')$; **while** $\overline{c_i} > 0$ { $\texttt{dec}(\overline{c_i})$; $\texttt{inc}(c'')$ }; $\texttt{inc}(\overline{c_i})$;
while $c'' > 0$
{ $\texttt{dec}(c'')$; **while** $\overline{c_i} > 0$ { $\texttt{dec}(\overline{c_i})$; $\texttt{inc}(c''')$ };
while $c''' > 0$ { $\texttt{dec}(c''')$; $\texttt{inc}(\overline{c_i})$; $\texttt{inc}(\overline{c_i})$ } } }

Fig. 3. Computing $2 \Uparrow m$.

Since $\preceq$ is a well-quasi-ordering, the procedure terminates and computes a basis of the set of all levels from which every sequence of transitions is finite, as required.

We shall establish that nonemptiness of safety ATRA_1 is not elementary by a two-stage reduction, which separates dealing with the inability of one-way alternating 1-register automata to detect incrementing errors in encodings of computations of counter machines, from ensuring acceptance only of encodings of computations in which counters are bounded by a tower of exponentiations. More precisely, we shall use the following problem as intermediary. The notation $2 \Uparrow m$ is as in Example 2.4.

(***) Given a deterministic counter machine $\mathcal{C}$ and $m \geq 1$ in unary, does $\mathcal{C}$ have a computation which possibly contains incrementing errors, in which every counter value is at most $2 \Uparrow m$, and which is either halting or infinite?

Such a machine is a tuple $\langle Q, q_I, q_H, k, \delta \rangle$ where: Q is a finite set of states, q_I is the initial state, q_H is the halting state, $k \in \mathbb{N}$ is the number of counters, and $\delta : Q \setminus \{q_H\} \to \{1, \ldots, k\} \times (Q \cup Q^2)$ is a transition function. Thus, from a state $q \neq q_H$, either $\delta(q)$ is of the form $\langle c, q' \rangle$, which means that the machine increments c and goes to q', or $\delta(q)$ is of the form $\langle c, q', q'' \rangle$, which means that, if c is zero, then the machine goes to q', else it decrements c and goes to q''. More precisely, a configuration is a state together with a counter valuation, and we write $\langle q, v \rangle \to \langle q', v' \rangle$ iff, for some $v_\surd \geq v$ and $v'_\surd \leq v'$,

—either $\delta(q) = \langle c, q' \rangle$ and $v'_\surd = v_\surd[c \mapsto v_\surd(c) + 1]$,
—or $\delta(q) = \langle c, q', q'' \rangle$, $v_\surd(c) = 0$ and $v'_\surd = v_\surd$,
—or $\delta(q) = \langle c, q'', q' \rangle$ and $v'_\surd = v_\surd[c \mapsto v_\surd(c) - 1]$.

We say that the transition is error-free iff the above holds with $v_\surd = v$ and $v'_\surd = v'$. A computation is a sequence $\langle q_0, v_0 \rangle \to \langle q_1, v_1 \rangle \to \cdots$ such that $q_0 = q_I$ and $v = \mathbf{0}$.

To show that (***) is not elementary, we reduce from the problem of whether a deterministic 2-counter machine of size m has an error-free halting computation of length at most $2 \Uparrow m$. Given such a machine $\mathcal{C}$ whose counters are c_1 and c_2, let $\widehat{\mathcal{C}}$ be a deterministic machine with counters c_1, c_2, $\overline{c_1}$, $\overline{c_2}$, $c^\dagger$, c', c'' and c''', which performs the following and then halts:

(I) For both $i \in \{1, 2\}$, set $\overline{c_i}$ to $2 \Uparrow m$ by executing the pseudo-code in Figure 3. The loops over c', c'' and c''' implement $\overline{c_i} := 2 \Uparrow c'$, $\overline{c_i} := 2^{c''}$ and $\overline{c_i} := 2 \times c'''$ (respectively).
(II) Simulate $\mathcal{C}$ using c_1 and c_2, and after each step:
—increment $c^\dagger$;
—if c_i has been incremented, then decrement $\overline{c_i}$;
—if c_i has been decremented, then increment $\overline{c_i}$;
—if $\mathcal{C}$ has halted, then go to (III).
(III) For both $i \in \{1, 2\}$, transfer $\overline{c_i}$ to c_i.

Observe that $\widehat{\mathcal{C}}$ is computable in space logarithmic in m. If $\mathcal{C}$ has an error-free halting computation of length at most $2 \Uparrow m$, running $\widehat{\mathcal{C}}$ without errors indeed halts and does not involve counter values greater than $2 \Uparrow m$. For the converse, suppose $\widehat{\mathcal{C}}$ has a computation which possibly contains incrementing errors, in which every counter value is at most $2 \Uparrow m$, and that is either halting or infinite. By the construction of $\widehat{\mathcal{C}}$ and the boundedness of counter values, the computation cannot be infinite, so it is halting. Since $\overline{c_1}$ and $\overline{c_2}$ were set to $2 \Uparrow m$ by stage (I), and since stage (III) terminated, the halting computation of $\mathcal{C}$ in stage (II) must have been error-free and it is certainly of length at most $2 \Uparrow m$.

To reduce from (***) to nonemptiness of safety ATRA_1, consider a deterministic counter machine $\mathcal{C} = \langle Q, q_I, q_H, k, \delta \rangle$ and $m \geq 1$. We can assume that $q' \neq q''$ whenever $\delta(q) = \langle c, q', q'' \rangle$. By the proof of [Demri and Lazić 2009, Theorem 5.2], which uses essentially the same encoding of computations of counter machines into data words as in the proof of Bojańczyk et al. [2006, Theorem 14], we have that an ATRA_1 $\mathcal{A}_{\mathcal{C}}$ with alphabet Q is computable in space logarithmic in $|\mathcal{C}|$, such that it safety-accepts a data tree τ iff the leftmost path in τ (that is, the sequence of nodes obtained by starting from the root and repeatedly taking the left-hand child) satisfies the following:

—the letter of the first node is q_I, and either the letter of the last nonleaf node is q_H or the sequence is infinite;
—for all letters q and q' of two consecutive nodes n and n' (respectively),
  —either $\delta(q)$ is of the form $\langle c, q' \rangle$ and we say that n is c-decrementing,
  —or $\delta(q)$ is of the form $\langle c, q', q'' \rangle$ and we say that n is c-zero-testing,
  —or $\delta(q)$ is of the form $\langle c, q'', q' \rangle$ and we say that n is c-decrementing;
—for each counter c, no two c-incrementing nodes are labeled by the same datum, no two c-decrementing nodes are labeled by the same datum, and whenever a c-incrementing node n is followed by a c-zero-testing node n', then a c-decrementing node with the same datum as n must occur between n and n'.

Hence, by taking the leftmost paths in data trees that are safety-accepted by $\mathcal{A}_{\mathcal{C}}$ and erasing data, we obtain exactly the sequences of states of halting or infinite computations of $\mathcal{C}$ that possibly contain incrementing errors. Assuming that $b_1, \ldots, b_m, *$ are not in Q, to restrict further to computations of $\mathcal{C}$ in which every counter value is at most $2 \Uparrow m$, it suffices to strengthen $\mathcal{A}_{\mathcal{C}}$ to obtain a safety ATRA_1 $\mathcal{A}_{\mathcal{C}}^{2 \Uparrow m}$ with alphabet $Q \cup \{b_1, \ldots, b_m, *\}$ which requires that:

—whenever a node n in the leftmost path is c-incrementing, then the automaton $\mathcal{B}_m$ from Example 2.4 safety-accepts the right-hand subtree at n;
—whenever a node n in the leftmost path is c-incrementing, n' is either n or a subsequent c-incrementing node, and no c-decrementing node with the same datum as n occurs between n and n', then the right-hand subtree at n' contains a node with letter b_m and the same datum as n.

Finally, that nonuniversality of safety ATRA_1 is not primitive recursive follows from the same lower bound for nonuniversality of safety one-way alternating automata with 1 register over data words [Lazić 2006]. □

5. XPATH SATISFIABILITY

In this section, we first describe how XML documents and DTDs can be represented by data trees and tree automata. We then introduce a forward fragment of XPath, and a safety subfragment. By translating XPath queries to forward alternating tree 1-register automata, and applying results from Sections 3 and 4, we obtain decidability

of satisfiablity for forward XPath on finite documents and for safety forward XPath on finite or infinite documents.

XML Trees. Suppose Σ is a finite set of element types, Σ' is a finite set of attribute names, and Σ and Σ' are disjoint. An XML document [Bray et al. 1998] is an unranked ordered tree whose every node n is labeled by some $\mathtt{type}(n) \in \Sigma$ and by a datum for each element of some $\mathtt{atts}(n) \subseteq \Sigma'$. Motivated by processing of XML streams (cf., e.g., Olteanu et al. [2004]), we do not restrict our attention to finite XML documents.

Concerning the data in XML documents, we shall consider only the equality predicate between data labels. Equality comparisons with constants are straightforward to encode using additional attribute names. Therefore, similarly as Bojańczyk et al. [2009], we represent an XML document by a data tree with alphabet $\Sigma \cup \Sigma'$, where each node n is represented by a sequence of $1 + |\mathtt{atts}(n)|$ nodes: the first node is labeled by $\mathtt{type}(n)$, the labels of the following nodes enumerate $\mathtt{atts}(n)$, the children of the last node represent the first child and the next sibling of n (if any), and for each preceding node in the sequence, its left-hand child is the next node and its right-hand child is a leaf. We say that such a data tree is an *XML tree*.

Following Benedikt et al. [2008] and Bojańczyk et al. [2009], we assume without loss of generality that document type definitions (DTDs) [Bray et al. 1998] are given as regular tree languages. More precisely, we consider a DTD to be a forward nondeterministic tree automaton $\mathcal{T}$ with alphabet $\Sigma \cup \Sigma'$ and without ε-transitions. Such automata can be defined by omitting counters and ε-transitions from ITCA (cf. Section 2.3). Infinite trees are processed in safety mode, that is, the condition that an infinite run of $\mathcal{T}$ has to satisfy to be accepting is the same as for finite runs: for each leaf n, the state of the configuration at n is final. An XML tree τ as above is regarded to satisfy $\mathcal{T}$ iff $\mathcal{T}$ accepts $\mathtt{tree}(\tau)$.

Fragments of XPath. The fragment of XPath [Clark and DeRose 1999] in this section contains all operators commonly found in practice and was considered in Benedikt et al. [2008] and Geerts and Fan [2005]. The grammars of queries p and qualifiers u are mutually recursive. The element types a and attribute names a' range over Σ and Σ', respectively.

$$p ::= \varepsilon \mid \triangledown \mid \triangle \mid \triangleright \mid \triangleleft \mid \triangledown^* \mid \triangle^* \mid \triangleright^* \mid \triangleleft^* \mid p/p \mid p \cup p \mid p[u]$$
$$u ::= \neg u \mid u \wedge u \mid p? \mid a \mid p/@a' = p/@a' \mid p/@a' \neq p/@a'$$

We say that a query or qualifier is *forward* iff:

—it does not contain $\triangle$, $\triangleleft$, $\triangle^*$ or $\triangleleft^*$;
—for every subqualifier of the form $p_1/@a'_1 \bowtie p_2/@a'_2$, we have that $p_1 = \varepsilon$ and that p_2 is of the form ε or $\triangledown/p'_2$ or $\triangleright/p'_2$.

A *safety* (respectively, *co-safety*) query or qualifier is one in which each occurence of $\triangledown$, $\triangledown^*$ or $\triangleright^*$ is under an odd (respectively, even) number of negations. Since infinite XML documents may contain infinite siblinghoods, $\triangledown$, $\triangledown^*$ and $\triangleright^*$ are exactly the queries that may require existence of a node which can be unboundedly far.

The semantics of queries and qualifiers is standard (cf., e.g., Geerts and Fan [2005]). We write the satisfaction relations as $\tau, n, n' \models p$ and $\tau, n \models u$, where τ is an XML tree $\langle N, \Sigma \cup \Sigma', \Lambda, \Delta \rangle$, and n and n' are Σ-labeled nodes. The definition is recursive over the grammars of queries and qualifiers, and can be found in Figure 4. We omit the Boolean cases, and we write $\widehat{\triangledown}$, $\widehat{\triangle}$, $\widehat{\triangleright}$ and $\widehat{\triangleleft}$ for the relations between Σ-labeled nodes that correspond to the child, parent, next-sibling and previous-sibling relations (respectively) in the document that τ represents.

We say that τ satisfies p iff $\tau, \varepsilon, n' \models p$ for some n'.

$$
\begin{aligned}
\tau, n, n' \models \varepsilon &\stackrel{\text{def}}{\Leftrightarrow} n = n' \\
\tau, n, n' \models \{\triangledown, \vartriangle, \triangleright, \triangleleft\} &\stackrel{\text{def}}{\Leftrightarrow} n\{\widehat{\triangledown}, \widehat{\vartriangle}, \widehat{\triangleright}, \widehat{\triangleleft}\}n' \\
\tau, n, n' \models \{\triangledown^*, \vartriangle^*, \triangleright^*, \triangleleft^*\} &\stackrel{\text{def}}{\Leftrightarrow} n\{\widehat{\triangledown}^*, \widehat{\vartriangle}^*, \widehat{\triangleright}^*, \widehat{\triangleleft}^*\}n' \\
\tau, n, n' \models p_1/p_2 &\stackrel{\text{def}}{\Leftrightarrow} \text{there exists } n'' \text{ such that} \\
&\qquad \tau, n, n'' \models p_1 \text{ and } \tau, n'', n' \models p_2 \\
\tau, n, n' \models p_1 \cup p_2 &\stackrel{\text{def}}{\Leftrightarrow} \tau, n, n' \models p_1 \text{ or } \tau, n, n' \models p_2 \\
\tau, n, n' \models p[u] &\stackrel{\text{def}}{\Leftrightarrow} \tau, n, n' \models p \text{ and } \tau, n' \models u \\
\tau, n \models p? &\stackrel{\text{def}}{\Leftrightarrow} \text{there exists } n' \text{ such that } \tau, n, n' \models p \\
\tau, n \models a &\stackrel{\text{def}}{\Leftrightarrow} \Lambda(n) = a \\
\tau, n \models p_1/@a'_1 \bowtie p_2/@a'_2 &\stackrel{\text{def}}{\Leftrightarrow} \text{there exist } n_1, k_1, n_2, k_2 \text{ such that} \\
&\qquad \tau, n, n_1 \models p_1, k_1 \leq |\mathtt{atts}(n_1)|, \Lambda(n_1 \cdot 0^{k_1}) = a'_1, \\
&\qquad \tau, n, n_2 \models p_2, k_2 \leq |\mathtt{atts}(n_2)|, \Lambda(n_2 \cdot 0^{k_2}) = a'_2, \\
&\qquad \Delta(n_1 \cdot 0^{k_1}) \bowtie \Delta(n_2 \cdot 0^{k_2})
\end{aligned}
$$

Fig. 4. Semantics of Queries and Qualifiers.

Example 5.1. Suppose $a'_1, a'_2 \in \Sigma'$. The forward query $p_{a'_1,a'_2} = \triangleright^*/\triangledown^*[\varepsilon/@a'_1 = (\triangledown/\triangledown^*)/@a'_2]$ is satisfied by Σ-labeled nodes n_0 and n_1 iff $n_0\widehat{\triangleright}^*\widehat{\triangledown}^*n_1$ and there exists n_2 such that $n_1\widehat{\triangledown}^+n_2$ and the value of attribute a'_1 at n_1 is equal to the value of attribute a'_2 at n_2. Hence, the safety forward query $\varepsilon[\neg(p_{a'_1,a'_2}?)]$ is satisfied by an XML tree over Σ and Σ' (whose root may have younger siblings) iff the value of a'_1 at a node is never equal to the value of a'_2 at a descendant.

Suppose a query p and a DTD $\mathcal{T}$ are over the same element types and attribute names. We say that p is satisfiable relative to $\mathcal{T}$ iff there exists an XML tree which satisfies p and $\mathcal{T}$. Finitary satisfiability restricts to finite XML trees.

Complexity of Satisfiability. Let us regard a forward qualifier u over element types Σ and attribute names Σ' as finitely equivalent to an ATRA_1 $\mathcal{A}$ with alphabet $\Sigma \cup \Sigma'$ iff, for every finite XML tree τ over Σ and Σ', and Σ-labeled node n, we have $\tau, n \models u$ iff $\mathcal{A}$ accepts the subtree of τ rooted at n. For safety (respectively, co-safety) u, safety (respectively, co-safety) equivalence is defined by also considering infinite XML trees and safety (respectively, co-safety) acceptance by $\mathcal{A}$.

To formalize the corresponding notions for queries, we introduce the following kind of automata "with holes". *Query automata* are defined in the same way as ATRA_1 (cf. Section 2.2), except that:

—transition formulae may contain a new atomic formula H;
—no path in the successor graph from the initial state to a state q such that H occurs in some transition formula at q may contain an update edge.

The vertices of the successor graph are all states, there is an edge from q to r iff $r(0, \downarrow)$, $r(0, \not\downarrow)$, $r(1, \downarrow)$ or $r(1, \not\downarrow)$ occurs in some transition formula at q, and such an edge is called update iff $r(0, \downarrow)$ or $r(1, \downarrow)$ occurs in some transition formula at q.

To define a run of a query automaton on a data tree τ with the same alphabet and with respect to a set of nodes N', we augment the definition of runs of ATRA_1 so that whenever a transition formula is evaluated at a node n, each occurence of H is treated as $\top$ if $n \in N'$, and as $\bot$ if $n \notin N'$. Acceptance of a finite data tree, safety acceptance,

and co-safety acceptance, all with respect to a set of nodes for interpreting H, are then defined as for ATRA_1.

For a query automaton $\mathcal{A}$ and an ATRA_1 or query automaton $\mathcal{A}'$ with the same alphabet and initial states q_I and q'_I (respectively), we define the substitution of $\mathcal{A}'$ for the hole in $\mathcal{A}$ by forming a disjoint union of $\mathcal{A}$ and $\mathcal{A}'$, taking q_I as the initial state, and substituting each occurence of H in each transition formula $\delta(q, a, b)$ of $\mathcal{A}$ by $\delta(q'_I, a, b)$. Observe that the unreachability in $\mathcal{A}$ of H from q_I by a path with an update edge means that the composite automaton transmits initial register values to $\mathcal{A}'$ without changes.

Now, we say that a forward query p over element types Σ and attribute names Σ' is finitely equivalent to a query automaton $\mathcal{B}$ with alphabet $\Sigma \cup \Sigma'$ iff, for every finite XML tree τ over Σ and Σ', Σ-labeled node n, and set N' of Σ-labeled nodes, we have $\tau, n, n' \models p$ for some $n' \in N'$ iff $\mathcal{B}$ accepts the subtree of τ rooted at n with respect to N'. For safety (respectively, co-safety) p, safety (respectively, co-safety) equivalence is defined by also considering infinite XML trees and safety (respectively, co-safety) acceptance by $\mathcal{B}$.

THEOREM 5.2. *For each forward query p (respectively, forward qualifier u) over Σ and Σ', a finitely equivalent query automaton $\mathcal{B}_p^{\Sigma,\Sigma'}$ (respectively, $ATRA_1$ $\mathcal{A}_u^{\Sigma,\Sigma'}$) is computable in logarithmic space. If p (respectively, u) is safety, then it is safety equivalent to $\mathcal{B}_p^{\Sigma,\Sigma'}$ (respectively, $\mathcal{A}_u^{\Sigma,\Sigma'}$).*

PROOF. The translations are defined recursively over the grammars of queries and qualifiers:

—$\mathcal{B}_\varepsilon^{\Sigma,\Sigma'}$, $\mathcal{B}_\triangledown^{\Sigma,\Sigma'}$, $\mathcal{B}_\triangleright^{\Sigma,\Sigma'}$, $\mathcal{B}_{\triangledown*}^{\Sigma,\Sigma'}$, $\mathcal{B}_{\triangleright*}^{\Sigma,\Sigma'}$ and $\mathcal{A}_a^{\Sigma,\Sigma'}$ are straightforward to define;
—$\mathcal{B}_{p\cup p'}^{\Sigma,\Sigma'}$ is formed from $\mathcal{B}_p^{\Sigma,\Sigma'}$ and $\mathcal{B}_{p'}^{\Sigma,\Sigma'}$ by disjunctive disjoint union, $\mathcal{A}_{\neg u}^{\Sigma,\Sigma'}$ is formed from $\mathcal{A}_u^{\Sigma,\Sigma'}$ by dualisation, and $\mathcal{A}_{u\wedge u'}^{\Sigma,\Sigma'}$ is formed from $\mathcal{A}_u^{\Sigma,\Sigma'}$ and $\mathcal{A}_{u'}^{\Sigma,\Sigma'}$ by conjunctive disjoint union (cf. the proof of Proposition 2.2);
—to obtain $\mathcal{B}_{p/p'}^{\Sigma,\Sigma'}$, we substitute $\mathcal{B}_{p'}^{\Sigma,\Sigma'}$ for the hole in $\mathcal{B}_p^{\Sigma,\Sigma'}$;
—to obtain $\mathcal{B}_{p[u]}^{\Sigma,\Sigma'}$, we substitute a conjunctive disjoint union of $\mathcal{B}_\varepsilon^{\Sigma,\Sigma'}$ and $\mathcal{A}_u^{\Sigma,\Sigma'}$ for the hole in $\mathcal{B}_p^{\Sigma,\Sigma'}$;
—$\mathcal{A}_{p?}^{\Sigma,\Sigma'}$ is formed from $\mathcal{B}_p^{\Sigma,\Sigma'}$ by substituting $\top$ for H;
—an automaton for $\varepsilon/@a'_1 = (\triangledown/p)/@a'_2$ is formed by substituting the second automaton depicted in Figure 5 (cf. Example 2.4 for depicting conventions) for the hole in $\mathcal{B}_p^{\Sigma,\Sigma'}$, and substituting the result for the hole in the first automaton depicted in Figure 5;
—the remaining cases in the grammar of qualifiers are handled similarly.

The required equivalences, as well as that if p (respectively, u) is co-safety then it is co-safety equivalent to $\mathcal{B}_p^{\Sigma,\Sigma'}$ (respectively, $\mathcal{A}_u^{\Sigma,\Sigma'}$), are shown simultaneously by induction. □

THEOREM 5.3.

(a) For forward XPath and arbitrary DTDs, satisfiability over finite XML trees is decidable.
(b) For safety forward XPath and arbitrary DTDs, satisfiability over finite or infinite XML trees is decidable.

PROOF. Given a forward query p and a DTD $\mathcal{T}$ over element types Σ and attribute names Σ', by Theorem 5.2, an ATRA_1 $\mathcal{A}_{p?}^{\Sigma,\Sigma'}$ is computable, which is finitely equivalent to the qualifier $p?$. We can then compute an ITCA $\mathcal{C}(\mathcal{A}_{p?}^{\Sigma,\Sigma'})$ as in the proof of Theorem 3.1,

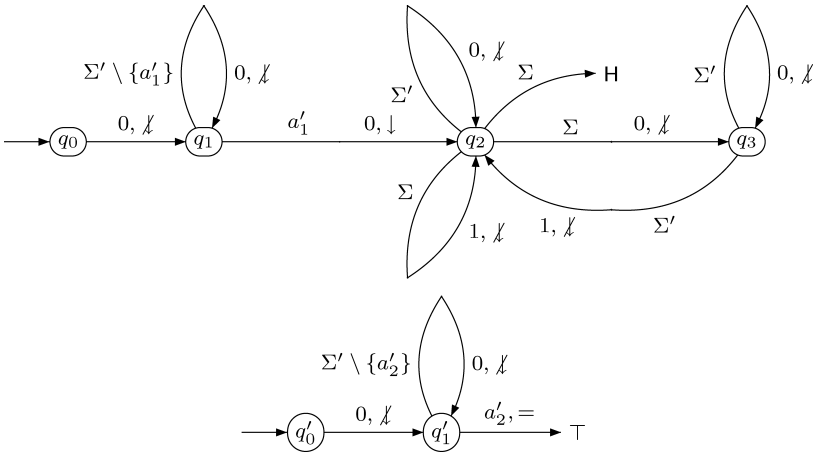

Fig. 5. Defining $\mathcal{A}^{\Sigma,\Sigma'}_{\varepsilon/@a'_1=(\nabla/p)/@a'_2}$.

which recognises exactly trees obtained by erasing data from finite XML trees that satisfy p. To conclude (a), we observe that ITCA are closed (in logarithmic space) under intersections with forward nondeterministic tree automata, and apply Theorem 2.5.

For (b), supposing that p is safety, by Theorem 5.2 again, an ATRA_1 $\mathcal{A}^{\Sigma,\Sigma'}_{p?}$ is computable, which is safety equivalent to the qualifier $p?$. Applying the proof of Theorem 4.1 to $\mathcal{A}^{\Sigma,\Sigma'}_{p?}$ and an ATRA_1 whose safety language is empty, we can compute an ITCANT $\mathcal{C}'(\mathcal{A}^{\Sigma,\Sigma'}_{p?})$, which contains no cycles of ε-transitions and recognises exactly trees obtained by erasing data from finite or infinite XML trees that satisfy p. It remains to observe that ITCANT with no cycles of ε-transitions are closed (in logarithmic space) under intersections with forward nondeterministic tree automata, and to recall that their nonemptiness was shown decidable also in the proof of Theorem 4.1. □

We remark that, by the proof of Demri and Lazić [2009, Theorem 5.2], finitary satisfiability for forward XPath with DTDs is not primitive recursive, even without sibling axes (that is, $\triangleright$ and $\triangleright^*$).

6. CONCLUDING REMARKS

It would be interesting to know more about the complexities of nonemptiness for safety ATRA_1 and satisfiability for safety forward XPath with DTDs. By Theorem 4.1, the former is decidable and not elementary, and by Theorem 5.3(b), the latter is decidable.

ACKNOWLEDGMENTS

We are grateful to the referees for helping us improve the presentation.

REFERENCES

Alpern, B. and Schneider, F. B. 1987. Recognizing safety and liveness. *Distr. Comput. 2,* 3, 117–126.

Benedikt, M., Fan, W., and Geerts, F. 2008. XPath satisfiability in the presence of DTDs. *J. ACM 55,* 2.

Björklund, H. and Bojańczyk, M. 2007. Bounded depth data trees. In *Proceeding of the 34th International Colloquium on Automata, Language and Programming (ICALP). Lecture Notes in Computer Science*, vol. 4596. Springer, 862–874.

Björklund, H. and Schwentick, T. 2007. On notions of regularity for data languages. In *In Proceeding of the 16th International Symposium on Fundamentals of Computation Theory (FCT). Lecture Notes in Computer Science*, vol. 4639. Springer, 88–99.

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

Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., and David, C. 2006. Two-variable logic on words with data. *In Proceeding of the 21st IEEE Symposiam on Login in Computer (LICS)*. IEEE Computer Society, 7–16.

Bray, T., Paoli, J., and Sperberg-McQueen, C. 1998. Extensible markup language (XML) 1.0. W3C Recommendation.

Brzozowski, J. A. and Leiss, E. L. 1980. On equations for regular languages, finite automata, and sequential networks. *Theoret. Comput. Sci. 10,* 1, 19–35.

Clark, J. and DeRose, S. 1999. XML path language (XPath). W3C Recommendation.

David, C. 2004. Mots et données infinies. M.S. thesis, Laboratoire d'Informatique Algorithmique: Fondements et Applications, Paris.

deGroote, P., Guillaume, B., and Salvati, S. 2004. Vector addition tree automata. In *Prceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 64–73.

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

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

Finkel, A. and Schnoebelen, P. 2001. Well-structured transitions systems everywhere! *Theoret. Comput. Sci. 256,* 1–2, 63–92.

Geerts, F. and Fan, W. 2005. Satisfiability of XPath queries with sibling axes. In *Proceedings of the 10th International Symposium on Database Programming Languages (DBPL)*. Lecture Notes in Computer Science, vol. 3774. Springer, 122–137.

Hallé, S., Villemaire, R., and Cherkaoui, O. 2006. CTL model checking for labeled tree queries. In *Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME)*. IEEE Comput. Society, 27–35.

Higman, G. 1952. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. 3,* 2, 7, 326–336.

Jurdziński, M. and Lazić, R. 2007. Alternation-free modal mu-calculus for data trees. In *Proceedings of the 22nd IEEE Symposium on Logic in Computing Science (LICS)*. IEEE Computer Society, 131–140.

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

Kaminski, M. and Tan, T. 2008. Tree automata over infinite alphabets. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occassion of His 85th Birthday*. Lecture Notes in Computer Science, vol. 4800. Springer, 386–423.

Lazić, R. 2006. Safely freezing LTL. In *Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science.* Lecture Notes in Computer Science, vol. 4337. Springer, 381–392.

Löding, C. and Thomas, W. 2000. Alternating automata and logics over infinite words. In *Proceedings of the IFIP International Conference on Theoretical Computer Science (IFIPTCS)*. Lecture Notes in Computer Science, vol. 1878. Springer, 521–535.

Muller, D. E., Saoudi, A., and Schupp, P. E. 1986. Alternating automata, the weak monadic theory of the tree, and its complexity. In *Proceedings of the 13th International Colloquinm on Automata, Languages and Programming (ICALP)*. Lecture Notes in Computer Science, vol. 226. Springer, 275–283.

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

Olteanu, D., Furche, T., and Bry, F. 2004. An efficient single-pass query evaluator for XML data streams. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*. ACM, 627–631.

Sakamoto, H. and Ikeda, D. 2000. Intractability of decision problems for finite-memory automata. *Theoret. Comput. Sci. 231,* 2, 297–308.

Segoufin, L. 2006. Automata and logics for words and trees over an infinite alphabet. In *Proceedings of the 20th International Workshop on Computer Science Logic (CSL). Lecture Notes in Computer Scince,* vol. 4207. Springer, 41–57.

Received May 2008; revised March 2010; accepted June 2010