Forward-XPath and extended register automata on data-trees (with Appendix)

Diego Figueira
INRIA, LSV,
ENS Cachan, France

ABSTRACT

We consider a fragment of XPath named 'forward-XPath', which contains all descendant and rightwards sibling axes as well as data equality and inequality tests. The satisfiability problem for forward-XPath in the presence of DTDs and even of primary key constraints is shown here to be decidable.

To show decidability we introduce a model of alternating automata on data trees that can move downwards and rightwards in the tree, have one register for storing data and compare them for equality, and have the ability to (1) nondeterministically guess a data value and store it, and (2) quantify universally over the set of data values seen so far during the run. This model extends the work of Jurdziński and Lazić. Decidability of the finitary non-emptiness problem for this model is obtained by a direct reduction to a well-structured transition system, contrary to previous approaches.

Categories and Subject Descriptors

I.7.2 [**Document Preparation**]: Markup Languages
; H.2.3 [**Database Management**]: Languages
; H.2.3 [**Languages**]: Query Languages

General Terms

Algorithms, Languages

Keywords

alternating tree register automata, XML, forward XPath, unranked ordered tree, data-tree, infinite alphabet

1. INTRODUCTION

This work is motivated by the increasing importance of reasoning tasks in XML research. An XML document can be seen as an unranked ordered tree where each node carries a *label* from a finite alphabet and a set of *attributes*, each with an associated datum from some infinite domain.

XPath is arguably the most widely used XML node selecting language, part of XQuery and XSLT; it is an open standard and a W3C Recommendation [5]. Static analysis on XML languages is crucial for query optimization tasks, consistency checking of XML specifications, type checking transformations, or many applications on security. Among the most important problems are those of query equivalence and query containment. By answering these questions we can decide at compile time whether the query contains a contradiction, and thus whether the computation of the query on the document can be avoided, or if one query can be safely replaced by another one. For logics closed under boolean combination, these problems reduce to satisfiability checking, and hence we focus on this problem. Unfortunately, the satisfiability problem for XPath with *data tests* is undecidable, even when the data domain has no structure [11] (i.e., where the only data relation available is the test for equality or inequality). It is then natural to identify and study decidable expressive fragments. In this work we adopt an automata-theoretic approach to find such fragments. The main contributions can be summarized as follows.

- A new register automata model for XML is introduced. This is an extension of the model treated in [13] with a decidable finitary emptiness problem. The decidability proof we propose simplifies the previous approaches of [13, 6] and facilitates the pursuit and identification of decidable extensions. This is evidenced here by the introduction of two extensions that preserve decidability.
- The satisfiability for the 'forward' fragment of XPath with data test equalities and inequalities is shown to be decidable, even in the presence of DTDs and primary key constraints. This settles a natural question left from the work in [13], also mentioned in [8, 9]. As a consequence this also answers positively the open question raised in [1] on whether the downward fragment of XPath in the presence of DTDs is decidable.[1] In fact, we give a decision procedure for the satisfiability problem in the presence of any regular tree language, and we can therefore code the core of XML Schema (stripped of functional dependencies, except of unary primary keys) or Relax NG document types.

ICDT 2010, March 22–25, 2010, Lausanne, Switzerland.

[1] The same question on downward XPath but in the *absence* of DTDs was treated in [8].

Automata. The automata model we define is based on the ATRA model (for Alternating Tree Register Automata). It is a tree walking automaton with alternating control and one register to store and compare data values. This automaton can move downwards and rightwards over an unranked ordered tree with data. It is a decidable model that has been studied in [13] and corresponds to the extension to trees of the automaton over words of [6]. The proofs of decidability of these automata models are based on non-trivial reductions to a class of decidable counter automata with faulty increments. In the present work, decidability is directly shown by interpreting the semantics of the automaton in the theory of well quasi-orderings in terms of a well-structured transition system [10]. The object of this alternative proof is twofold. On the one hand, we propose a simpler proof of the main decidability results of [13, 6]. On the other, our approach easily yields the decidability of the non-emptiness problem for two powerful extensions of ATRA. These extensions consist in the following abilities: (a) the automaton can nondeterministically *guess* any data value of the domain and store it in the register; and (b) it can make a certain kind of universal quantification over the data values seen along the run of the automaton, in particular over the ancestors' data values. We name these extensions guess and spread respectively, and the model of alternating tree register automata with these extensions as ATRA(guess, spread). We show that this model of automata can decide a large fragment of XPath.

XML. We study and show decidability of the satisfiability problem for a fragment of XPath by a reduction to the non-emptiness problem of ATRA(guess, spread). Let us describe this logic. Core-XPath [12] is the fragment of XPath that captures all the navigational behavior of XPath. It has been well studied and its satisfiability problem is known to be decidable in EXPTIME in the presence of DTDs [14]. We consider an extension of this language with the possibility to make *equality* and *inequality* tests between attributes of XML elements. This logic is named Core-Data-XPath in [2], and its satisfiability problem is undecidable [11]. The present work contributes to the study of different navigational fragments of XPath with equality tests in the attempt to find decidable and computationally well-behaved logics. Here we address a large fragment named 'forward XPath', that contains the child, descendant, self-or-descendant, next-sibling, following-sibling, and self-or-following-sibling axes. For economy of space we refer to these axes as $\downarrow$, $\downarrow^{+}$, $\downarrow^{*}$, $\rightarrow$, $\rightarrow^{+}$, $\rightarrow^{*}$ respectively. Note that $\rightarrow^{+}$ and $\rightarrow^{*}$ are interdefinable in the presence of $\rightarrow$, and similarly with $\downarrow^{+}$ and $\downarrow^{*}$. We then refer to this fragment as XPath($\downarrow, \downarrow^{*}, \rightarrow, \rightarrow^{*}$). Although our automata model cannot capture this logic in terms of expressiveness, we show that there is a non-trivial reduction to the nonemptiness problem of ATRA(guess, spread). By the fact that these automata can code any regular language (in particular a DTD), and that XPath($\downarrow, \downarrow^{*}, \rightarrow, \rightarrow^{*}$) can express unary primary key constraints, it follows that satisfiability of forward-XPath in the presence of DTDs and primary key constraints is decidable.

Related work

In [13] a fragment of XPath($\downarrow, \downarrow^{*}, \rightarrow, \rightarrow^{*}$) is treated. The language is restricted to data test formulæ of the form $\varepsilon = \alpha$ (or $\varepsilon \neq \alpha$), that is, sentences that test whether there exists an element accessible via the α-relation with the same (resp. different) data value as the *current* node of evaluation. This logic was shown to be expressible in the ATRA automaton defined in [13]. However, this restricted form of data tests cannot express, e.g., that there are two leaves with the same datum, or that all the elements with a certain symbol have different data value (i.e, a primary key constraint). The problem regarding the decidability of the *full* forward fragment with *arbitrary* data tests is a non-trivial natural question left from [13] that is positively answered here.

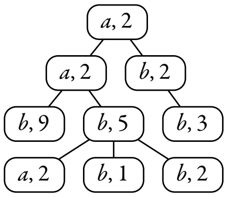


Figure 1: A data tree.

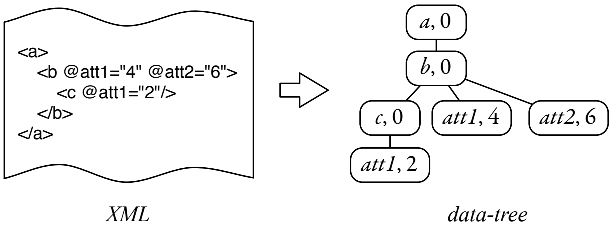
The work in [1] investigates the satisfiability problem for many XPath logics, mostly fragments without negation or without data equality tests in the absence of sibling axes. Also, in [8] there is a thorough study of the satisfiability problem for all the *downward* XPath queries with and without data equality tests. Notably, none of these works considers *horizontal* axes to navigate between siblings: By exploiting the bisimulation invariance property enjoyed by these logics, the complexity of the satisfiability problem is kept relatively low (at most EXPTIME) in the presence of data values. However, when horizontal axes are present, most of the problems have a non-primitive recursive complexity (including the fragment of [13], or even much simpler ones without the one-step '→' axis [9]). In [11], several fragments with horizontal axes are treated. The only fragment with data tests and negation studied there is incomparable with the forward fragment, and it is shown to be undecidable.

First-order logic with two variables and data equality tests is explored in [2], where it is shown that FO^2 with local one-step relations to move around the data tree and a data equality test relation is decidable. [2] also shows the decidability of a fragment of XPath($\uparrow, \downarrow, \leftarrow, \rightarrow$) with sibling and upward axes but restricted to local elements and to data formulæ of the kind $\varepsilon = \alpha$ (or $\neq$), while our fragment cannot move upwards but features transitive axes and unrestricted data tests.

2. DATA TREES AND XML DOCUMENTS

In this article we work with *data trees* instead of XML documents, being a simpler formalism to work with, from where results can be transferred to the class of XML documents. We discuss below how all the results we give on XPath over data trees, also hold for the class of XML documents.

A **data tree** is an unranked ordered tree whose every node is labeled by a symbol from a finite alphabet and a datum from an infinite domain, as in the example of Fig. 1. Let $\wp(S)$ denote the power set of S, let $\mathbb{N}$ be the set of positive integers, and let us fix $\mathbb{D}$ to be any infinite domain of data values. In our examples we will consider $\mathbb{D} = \mathbb{N}$. We define $\mathsf{Pos} \subseteq \wp(\mathbb{N}^*)$ to be the set of sets of *finite tree* positions



```
<a>
   <b @att1="4" @att2="6">
      <c @att1="2"/>
   </b>
</a>
```

Figure 2: From XML documents to data-trees.

(we write ϵ for the empty word, corresponding to the root's position). $X \in \mathsf{Pos}$ iff (a) $X \subseteq \mathbb{N}^*, |X| < \infty$; (b) it is prefix-closed; and (c) if $n(i+1) \in X$ then $ni \in X$. Given a finite alphabet Σ, a *finite data tree* over Σ is a tuple $\mathcal{T} = \langle P, \sigma \rangle$ with $P \in \mathsf{Pos}$ and $\sigma : P \to \Sigma \times \mathbb{D}$, as in Fig. 1. The functions π_1 and π_2 project the first and second component of an element of $\Sigma \times \mathbb{D}$. We define $\mathsf{type}_{\mathcal{T}} : P \to \{\triangledown, \bar{\triangledown}\} \times \{\triangleright, \bar{\triangleright}\}$ that specifies whether a node has children and/or siblings to the right. That is, $\mathsf{type}_{\mathcal{T}}(p) := (a, b)$ where $a = \triangledown$ iff $p1 \in P$, and where $b = \triangleright$ iff $p = p'i$ and $p'(i+1) \in P$.

While a data tree has one data value for each node, an XML document may have several attributes at a node, each with a data value. Every attribute of an XML element can be encoded as a child node in a data tree labeled by the attribute's name, as in Fig. 2. This coding can be enforced by the formalisms we present below, and we can thus transfer all the decidability results to the class of XML documents. In fact, it suffices to demand that all the attribute symbols can only occur at the leaves of the data tree and to interpret attribute expressions like '@*attrib1*' of XPath formulæ as child path expressions '$\mathsf{child}[attrib1]$'.

3. THE ATRA MODEL

In this section we present the model of computation that will enable us to show decidability of XPath and temporal logic fragments.

An **Alternating Tree Register Automaton** (ATRA) consists in a top-down walking automaton with alternating control and *one* register to store and test data. In [13] it was shown that its finitary emptiness problem is decidable and non primitive recursive. Here, we consider an extension of ATRA with two operators: spread and guess. We call this model ATRA(spread, guess).

Definition 3.1. *A forward alternating register automaton ATRA(spread, guess) is a tuple $\langle \Sigma, Q, q_I, \delta \rangle$ s.t.*

Σ is a finite alphabet; Q is a finite set of states; $q_I \in Q$ is the initial state; and

$\delta : Q \to \Phi$ is the transition function, where Φ is defined by the grammar

$$a \mid \bar{a} \mid \odot? \mid \mathsf{set}(q) \mid \mathsf{eq} \mid \overline{\mathsf{eq}} \mid q \wedge q' \mid q \vee q' \mid \triangledown q \mid \triangleright q \mid \mathsf{guess}(q) \mid \mathsf{spread}(q, q')$$

where $a \in \Sigma, q, q' \in Q$, $\odot \in \{\triangledown, \bar{\triangledown}, \triangleright, \bar{\triangleright}\}$.

This formalism without the guess and spread transitions is equivalent to the automata model of [13] on finite data trees, where $\triangledown$ and $\triangleright$ are to move to the first child or to the next sibling, $\mathsf{set}(q)$ stores the current datum and eq (resp. $\overline{\mathsf{eq}}$) tests that the current node's value is (resp. not) equal to the stored.

As this automaton is one-way, we define its semantics as a set of 'threads' for each node that progress synchronously. That is, all threads at a node move one step forward simultaneously and then perform some non-moving transitions independently. This is done for the sake of simplicity of the formalism, which simplifies the presentation of the decidability proof.

Next we define a *configuration* of a node and a configuration of a tree to then give a notion of a *run* over a data tree $\mathcal{T} = \langle P, \sigma \rangle$. A **node configuration** is a tuple $\langle p, \alpha, \gamma, H \rangle$ that describes the partial state of the execution at a given node. $p \in P$ is the *node position* in the tree $\mathcal{T}$, $\gamma = \sigma(p) \in \Sigma \times \mathbb{D}$ is the current node's symbol/datum, and $\alpha = \mathsf{type}_{\mathcal{T}}(p)$ is the *tree type* of the node. Finally, $H \in \wp(Q \times \mathbb{D})$ is a finite collection of active **threads** of execution, each thread $\langle q, d \rangle$ consisting in a state q and the value stored in the register d. By $\mathsf{Conf}_{\mathcal{N}}$ we denote the set of all node configurations. A **tree configuration** is a finite set of node configurations, like $\{\langle \epsilon, \alpha, \gamma, H \rangle, \langle 1211, \alpha', \gamma', H' \rangle, \dots\}$. The run will be defined in such a way that a tree configuration never contains node configurations in a descendant/ancestor relation. We call $\mathsf{Conf}_{\mathcal{T}} = \wp(\mathsf{Conf}_{\mathcal{N}})$ the set of all finite tree configurations. Given a set of threads we write $data(H) := \{d \mid \langle q, d \rangle \in H\}$, and $data(\langle p, \alpha, (a, d), H \rangle) := \{d\} \cup data(H)$.

To define a run we first introduce three transition relations over *node configurations*: the *non-moving* relation $\to_\varepsilon$, the *first-child* relation $\to_\triangledown$, and the *next-sibling* relation $\to_\triangleright$. We start with $\to_\varepsilon$. If the transition corresponding to a thread is a $\mathsf{set}(q)$, the automaton *sets* the register with current data value and continues the execution of the thread with state q; if it is eq, the thread *accepts* (and in this case disappears from the configuration) if the current datum is *equal* to that of the register, otherwise the computation for that thread cannot continue. The reader can check that the rest of the cases follow the intuition of an alternating automaton. Let $\rho = \langle p, \alpha, (s, d), \{\langle q, d' \rangle\} \cup H \rangle$. Then,

$$\rho \to_\varepsilon \langle p, \alpha, (s, d), \{\langle q_i, d' \rangle\} \cup H \rangle \quad \text{if } \delta(q) = q_1 \vee q_2, i \in \{1, 2\} \tag{1}$$

$$\rho \to_\varepsilon \langle p, \alpha, (s, d), \{\langle q_1, d' \rangle, \langle q_2, d' \rangle\} \cup H \rangle \quad \text{if } \delta(q) = q_1 \wedge q_2 \tag{2}$$

$$\rho \to_\varepsilon \langle p, \alpha, (s, d), \{\langle q', d \rangle\} \cup H \rangle \quad \text{if } \delta(q) = \mathsf{set}(q') \tag{3}$$

$$\rho \to_\varepsilon \langle p, \alpha, (s, d), H \rangle \quad \text{if } \delta(q) = \mathsf{eq} \text{ and } d = d' \tag{4}$$

$$\rho \to_\varepsilon \langle p, \alpha, (s, d), H \rangle \quad \text{if } \delta(q) = \overline{\mathsf{eq}} \text{ and } d \neq d' \tag{5}$$

$$\rho \to_\varepsilon \langle p, \alpha, (s, d), H \rangle \quad \text{if } \delta(q) = \ell? \text{ and } \ell \text{ is in } \alpha \tag{6}$$

$$\rho \to_\varepsilon \langle p, \alpha, (s, d), H \rangle \quad \text{if } \delta(q) = s \tag{7}$$

$$\rho \to_\varepsilon \langle p, \alpha, (s, d), H \rangle \quad \text{if } \delta(q) = \bar{r} \text{ for } r \neq s \tag{8}$$

The following cases correspond to our extensions to the model of [13]. The 'guess' instruction extends the model with the ability of storing *any* datum from the domain $\mathbb{D}$.

Whenever $\delta(q) = \mathsf{guess}(q')$ is executed, a data value (non-deterministically chosen) is saved in the register.

$$\rho \to_\varepsilon \langle p, \alpha, (s, d), \{\langle q', e\rangle\} \cup H\rangle \\ \text{if } \delta(q) = \mathsf{guess}(q'), e \in \mathbb{D} \tag{9}$$

The 'spread' instruction is an unconventional operator in the sense that it depends on the data of *all* threads in the current configuration with a certain state. Whenever $\delta(q) = \mathsf{spread}(q_2, q_1)$ is executed, a new thread with state q_1 and datum d is created for each thread $\langle q_2, d\rangle$ present in the configuration. With this operator we can code a universal quantification over all the ancestors' data values. For convenience, we demand that this transition may only be applied if all other possible $\to_\varepsilon$ kind of transitions were already executed. Or, in other words, that only spread transitions or *moving* transitions are present in the configuration (the moving transitions being those defined as '$\triangledown q$' and '$\triangleright q$').

$$\begin{aligned}\rho \to_\varepsilon \langle p, \alpha, (s, d), \{\langle q_1, d\rangle \mid \langle q_2, d\rangle \in H\} \cup H\rangle& \\ \text{if } \delta(q) = \mathsf{spread}(q_2, q_1) \text{ and}& \\ \text{for all } \langle \tilde{q}, \tilde{d}\rangle \in H :& \\ \text{either } \delta(\tilde{q}) = \mathsf{spread}(\tilde{q}_1, \tilde{q}_2),\ \delta(\tilde{q}) = \triangledown\tilde{q}_1,& \\ \text{or } \delta(\tilde{q}) = \triangleright\tilde{q}_1 \text{ for some } \tilde{q}_1, \tilde{q}_2 \in Q&\end{aligned} \tag{10}$$

The $\to_\triangledown$ and $\to_\triangleright$ transitions advance all threads of the node simultaneously, and are defined, for any type $\alpha_1 \in \{\triangledown, \bar{\triangledown}, \triangleright, \bar{\triangleright}\}$ and symbol and with data value $\gamma_1 \in \Sigma \times \mathbb{D}$,

$$\langle p, (\triangledown, r), \gamma, H\rangle \to_\triangledown \langle p\,1, \alpha_1, \gamma_1, H_\triangledown\rangle, \tag{11}$$

$$\langle pi, (l, \triangleright), \gamma, H\rangle \to_\triangleright \langle p(i+1), \alpha_1, \gamma_1, H_\triangleright\rangle \tag{12}$$

iff (i) the configuration is 'moving' (i.e., all the threads $\langle q, d\rangle$ contained in H are of the form $\delta(q) = \triangledown q'$ or $\delta(q) = \triangleright q'$); (ii) for $\odot \in \{\triangledown, \triangleright\}$, $H_\odot = \{\langle q', d\rangle \mid \langle q, d\rangle \in H, \delta(q) = \odot\, q'\}$; and (iii) α_1 and γ_1 are consistent with the position $p1$ in the case of (11), or with $p(i+1)$ in the case of (12).

Finally, we define the transition between tree configurations that we call $\overset{t}{\rightarrowtail}$. This corresponds to applying a 'non-moving' $\to_\varepsilon$ to a node configuration, or to apply a 'moving' $\to_\triangledown$, $\to_\triangleright$, or both to a node configuration according to its type. That is, we define $\mathcal{S}_1 \overset{t}{\rightarrowtail} \mathcal{S}_2$ iff one of the following conditions holds:

1. $\mathcal{S}_1 = \{\rho\} \cup \mathcal{S}'$, $\mathcal{S}_2 = \{\tau\} \cup \mathcal{S}'$, $\rho \to_\varepsilon \tau$;
2. $\mathcal{S}_1 = \{\rho\} \cup \mathcal{S}'$, $\mathcal{S}_2 = \{\tau\} \cup \mathcal{S}'$, $\rho = \langle p, (\triangledown, \bar{\triangleright}), \gamma, H\rangle$, $\rho \to_\triangledown \tau$;
3. $\mathcal{S}_1 = \{\rho\} \cup \mathcal{S}'$, $\mathcal{S}_2 = \{\tau\} \cup \mathcal{S}'$, $\rho = \langle p, (\bar{\triangledown}, \triangleright), \gamma, H\rangle$, $\rho \to_\triangleright \tau$;
4. $\mathcal{S}_1 = \{\rho\} \cup \mathcal{S}'$, $\mathcal{S}_2 = \{\tau_1, \tau_2\} \cup \mathcal{S}'$, $\rho = \langle p, (\triangledown, \triangleright), \gamma, H\rangle$, $\rho \to_\triangledown \tau_1$, $\rho \to_\triangleright \tau_2$.

A *run* over a data tree $\mathcal{T} = \langle P, \sigma\rangle$ is a non-empty sequence $\mathcal{S}_1 \overset{t}{\rightarrowtail} \cdots \overset{t}{\rightarrowtail} \mathcal{S}_n$ with $\mathcal{S}_1 = \{\langle \epsilon, \alpha_0, \gamma_0, H_0\rangle\}$ and $H_0 = \{\langle q_I, \pi_2(\sigma(\epsilon))\rangle\}$ (i.e., the thread consisting in the initial state with the root's datum), such that for every $i \in [1..n]$, $\langle p, \alpha, \gamma, H\rangle \in \mathcal{S}_i$: (1) $p \in P$; (2) $\gamma = \sigma(p)$; and (3) $\alpha = \mathsf{type}_{\mathcal{T}}(p)$. We say that the run is *accepting* iff

$$\mathcal{S}_n \subseteq \{\langle p, \alpha, \gamma, \emptyset\rangle \mid \langle p, \alpha, \gamma, \emptyset\rangle \in \mathsf{Conf}_{\mathcal{N}}\}.$$

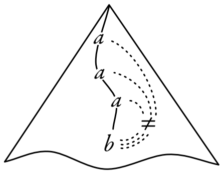


Figure 3: A property not expressible in ATRA.

The ATRA model is closed under all boolean operations [13]. However, the extensions introduced guess and spread, while adding expressive power, are not closed under complement as a trade-off for decidability.

Proposition 3.1. *(a) ATRA(guess) has more expressive power than ATRA; (b) ATRA(spread) has more expressive power than ATRA.*

Proof (sketch). (a) is based on the fact that while with guess we can express "*there are two leaves with equal datum*", with ATRA we cannot do it(more details in the Appendix). Here we focus on the proof of (b). We show a property $\mathcal{P}$ that can be expressed in ATRA(spread) and such that its negation permits to code an undecidable problem. Hence, $\mathcal{P}$ cannot be expressed in ATRA since it is both closed under complement and decidable.

Given $\Sigma = \{a, b\}$, let $\mathcal{P}$ be the following property: "*there exists an inner node labeled b such that there is no ancestor labeled a with the same data value*" as depicted in Fig. 3. Let us see how $\mathcal{P}$ can be coded into ATRA(spread). Assuming q_0 is the initial state, the transitions should reflect that every a seen along the run is saved with a state q_a, and that this state is in charge of propagating this datum everywhere in the tree. Then, we non-deterministically choose a b and check that all these stored values under q_a are different from the current one. For succinctness we write the transitions as positive boolean combinations of the basic operations.

$$\begin{aligned}\delta(q_0) = (b \wedge \triangledown? \wedge \mathsf{spread}(q_a, q_1)) \vee& \\ ((\bar{a} \vee \mathsf{set}(q_a)) \wedge (\bar{\triangledown}? \vee \triangledown q_0) \wedge (\bar{\triangleright}? \vee \triangleright q_0)),& \\ \delta(q_1) = \overline{\mathsf{eq}}, \qquad \delta(q_a) = (\bar{\triangledown}? \vee \triangledown q_a) \wedge (\bar{\triangleright}? \vee \triangleright q_a).&\end{aligned}$$

$\mathcal{P}$, on the other hand, cannot be expressed by the ATRA model. Were it expressible, then the negation "*for every inner node b there exists an ancestor a with the same data value*" would also be. It can be seen that with this kind of property one can code an accepting run of a Minsky automaton along a branch by using data to assure that (i) for every increment there is a corresponding future decrement; and by using this property that (ii) for every decrement there exists a corresponding previous increment. This is absurd, as the ATRA model is decidable. We refer the reader to [6, 9] for more details on these kind of codings. □

Proposition 3.2. *ATRA(spread, guess) models have the following properties: (i) they are closed under union, (ii) they are closed under intersection, (iii) they are not closed under complement.*

Proof (sketch). (i) and (ii) are straightforward if we notice that the first argument of spread ensures that this transition is always relative to the states of one of the automata being under intersection or union. (iii) is a consequence of the proof of Proposition 3.1 item (b), combined with the fact that the model will be shown to be decidable. □

3.1 Decidability of the emptiness problem

We dedicate this section to prove the decidability of the ATRA(guess, spread) emptiness problem. The main argument consists in interpreting the automaton's execution as a transition system in the theory of well quasi-orderings with some good properties that allow us to obtain an effective procedure for the emptiness problem. This is known in the literature as a well-structured transition system (WSTS) [10]. The following are standard definitions.

Definition 3.2. *$(\mathcal{A}, \leq)$ is a* **well quasi-order (wqo)** *iff '$\leq$' $\subseteq \mathcal{A} \times \mathcal{A}$ is a relation that is reflexive, transitive and for every infinite succession $w_1, w_2, \ldots \in \mathcal{A}^\omega$ there are two indexes $i < j$ such that $w_i \leq w_j$.*

Definition 3.3. *Given a transition system $(\mathcal{A}, \rightarrow)$, we define $Succ(a) := \{a' \mid a \rightarrow a'\}$, $Succ^*(a) := \{a' \mid a \rightarrow^* a'\}$. Given a wqo $(\mathcal{A}, \leq)$ and $\mathcal{A}' \subseteq \mathcal{A}$, we define $\uparrow \mathcal{A}' := \{a \mid a' \in \mathcal{A}', a' \leq a\}$.*

Definition 3.4. *We say that a transition system $(\mathcal{A}, \rightarrow)$ is* **finitely branching** *iff $Succ(a)$ is finite for all $a \in \mathcal{A}$. If $Succ(a)$ is also effectively computable for all a, we say that $(\mathcal{A}, \rightarrow)$ is* **effective**.

Definition 3.5. *A* wqo *$(\mathcal{A}, \leq)$ is* **reflexive downwards compatible (rdc)** *with respect to a transition system $(\mathcal{A}, \rightarrow)$ iff for every $a_1, a_2, a'_1 \in \mathcal{A}$ such that $a'_1 \leq a_1$ and $a_1 \rightarrow a_2$, there exists $a'_2 \in \mathcal{A}$ such that $a'_2 \leq a_2$ and either $a'_1 \rightarrow a'_2$ or $a'_1 = a'_2$.*

Decidability will be shown as a consequence of the following known result.

Proposition 3.3. ([10, Proposition 5.4]) *If $(\mathcal{A}, \leq)$ is a* wqo *and $(\mathcal{A}, \rightarrow)$ a transition system such that (1) it is* rdc*, (2) it is effective, and (3) $\leq$ is decidable; then for any $a \in \mathcal{A}$ it is possible to compute a finite set $\mathcal{A}' \subseteq \mathcal{A}$ such that $\uparrow \mathcal{A}' = \uparrow Succ^*(a)$.*

Theorem 3.1. *Non-emptiness of ATRA(guess, spread) is decidable.*

Proof. As already mentioned, decidability for ATRA was proved in [13]. Here we propose an alternative approach that simplifies the proof of decidability of the two extensions spread and guess.

The proof goes as follows. We will define a wqo $\prec$ over the *node configurations* and show that $(\mathsf{Conf}_\mathcal{N}, \prec)$ is rdc w.r.t. $\rightarrow_\varepsilon$, $\rightarrow_\triangledown$ and $\rightarrow_\triangleright$ (Lemma 3.2). We will then apply a useful result (Proposition 3.4) to lift this result to the set of tree configurations and prove for some decidable wqo $\sqsubset$ that $(\mathsf{Conf}_\mathcal{T}, \sqsubset)$ is rdc w.r.t. $\overset{t}{\rightarrowtail}$. Note that strictly speaking $\overset{t}{\rightarrowtail}$ is an infinite-branching transition system as $\rightarrow_\triangledown$ or $\rightarrow_\triangleright$ may take *any* value from the infinite set $\mathbb{D}$, and $\rightarrow_\varepsilon$ can also guess any value. However, it can trivially be restricted to an effective *finitely* branching one. Then, by Proposition 3.3, $\overset{t}{\rightarrowtail}$ has an effectively computable upward-closed reachability set, and this implies that the emptiness problem of ATRA(guess, spread) is decidable.

We first define the relation '$\prec$' $\subseteq \mathsf{Conf}_\mathcal{N} \times \mathsf{Conf}_\mathcal{N}$ between node configurations

$$\langle p, \alpha, (s, d), H\rangle \prec \langle p', \beta, (s', d'), H'\rangle$$

iff there exists an *injective* mapping $f : \{d\} \cup data(H) \rightarrow \mathbb{D}$ such that
1. if $\langle q, e\rangle \in H$ then $\langle q, f(e)\rangle \in H'$, 2. $f(d) = d'$, and 3. $s = s'$ and $\alpha = \beta$.

Whenever it is necessary to make explicit the witnessing function f that enables the relation, we write $\rho \prec_f \tau$. The following lemma follows from the definition just seen. The proof is in the Appendix.

Lemma 3.1. *$(\mathsf{Conf}_\mathcal{N}, \prec)$ is a well quasi-order.*

Let $\rightarrowtail \; := \; \rightarrow_\varepsilon \cup \rightarrow_\triangledown \cup \rightarrow_\triangleright \; \subseteq \mathsf{Conf}_\mathcal{N} \times \mathsf{Conf}_\mathcal{N}$. The core of this proof is centered in the following lemma.

Lemma 3.2. *$(\mathsf{Conf}_\mathcal{N}, \prec)$ is reflexive downward compatible (rdc) with respect to $\rightarrowtail$.*

Proof. We shall show that for all $\rho, \tau, \rho' \in \mathsf{Conf}_\mathcal{N}$ such that $\rho \rightarrowtail \tau$ and $\rho' \prec \rho$, there is τ' such that $\tau' \prec \tau$ and either $\rho' \rightarrowtail \tau'$ or $\tau' = \rho'$. The proof is a simple case analysis of the definitions for $\rightarrowtail$. All cases are treated alike, here we present the most representative. Suppose first that $\rightarrowtail$ performs a $\rightarrow_\varepsilon$, then one of the definition conditions of $\rightarrow_\varepsilon$ must apply.

If (4), let

$$\rho = \langle p, \alpha, (s, d), \{\langle q, d\rangle\} \cup H\rangle \rightarrowtail \tau = \langle p, \alpha, (s, d), H\rangle$$

with $\delta(q) = \mathsf{eq}$. Let $\rho' = \langle p', \alpha, (s, e'), H'\rangle \prec_f \rho$. If there is $\langle q, e\rangle \in H'$ such that $f(e) = d$, then by injectivity of f, $e = e'$ and we can then apply the same $\rightarrow_\varepsilon$-transition obtaining

$$\begin{tikzcd} \rho \arrow[r, phantom, "\rightarrow_\varepsilon"] \arrow[d, phantom, "\curlyvee"] & \tau \arrow[d, phantom, "\curlyvee"] \\ \rho' \arrow[r, phantom, "\rightarrow_\varepsilon"] & \tau' \end{tikzcd}$$

witenessed by the map f. Otherwise, we can safely take $\rho' = \tau'$ and check that $\tau' \prec_f \tau$.

If (3), let

$$\rho = \langle p, \alpha, (s, d), \{\langle q, d'\rangle\} \cup H\rangle \rightarrowtail$$
$$\tau = \langle p, \alpha, (s, d), \{\langle q', d\rangle\} \cup H\rangle$$

with $\rho \rightarrow_\varepsilon \tau$ and $\delta(q) = \mathsf{set}(q')$. Again let $\rho' \prec_f \rho$ containing $\langle q, e\rangle \in H'$ with $f(e) = d'$. In this case we can apply the same $\rightarrow_\varepsilon$-transition arriving to τ' where $\tau' \prec_f \tau$. Else, we take $\rho' = \tau'$.

If a guess is performed (9), let

$$\rho = \langle p, \alpha, (s, d), \{\langle q, d'\rangle\} \cup H\rangle \rightarrow_\varepsilon$$
$$\tau = \langle p, \alpha, (s, d), \{\langle q', e\rangle\} \cup H\rangle$$

with $\delta(q) = \mathsf{guess}(q')$. Let $\rho' = \langle p', \alpha, (s, d_1'), H'\rangle \prec_f \rho$. Suppose there is $\langle q, d_2'\rangle \in H'$ such that $f(d_2') = d'$, then we then take a guess transition from ρ' obtaining some τ'. If $e \in Im(f)$, we obtain τ' by guessing $f^{-1}(e)$ and hence $\tau' \prec_f \tau$. If $e \notin Im(f)$, τ' is obtained by guessing a 'new' value e_2 different from all those of $data(\rho')$, and by defining $f' := f[e_2 \mapsto e]$ we have $\tau' \prec_{f'} \tau$. Otherwise, if there is no $\langle q, d_2'\rangle \in H'$ such that $f(d_2') = d'$, we take $\tau' = \rho'$ and check that $\tau' \prec_f \tau$.

Finally, if a spread is performed (10), let

$$\rho = \langle p, \alpha, \gamma, \{\langle q, d'\rangle\} \cup H\rangle \rightarrow_\varepsilon$$
$$\tau = \langle p, \alpha, \gamma, \{\langle q_1, d\rangle \mid \langle q_2, d\rangle \in H\} \cup H\rangle$$

with $\delta(q) = \mathsf{spread}(q_2, q_1)$. Let $\rho' = \langle p', \alpha, \gamma', H'\rangle \prec_f \rho$ and suppose there is $\langle q, e\rangle \in H'$ such that $f(e) = d'$ (otherwise $\tau' = \rho'$ works). We then take a spread instruction $\rho' \rightarrow_\varepsilon \tau'$ and see that $\tau' \prec_f \tau$, because any $\langle q_1, e\rangle$ in τ' generated by the spread must come from $\langle q_2, e\rangle$ of ρ', and hence from some $\langle q_2, f(e)\rangle$ of ρ; now by the spread applied on ρ, $\langle q_1, f(e)\rangle$ is in τ. The remaining cases of $\rightarrow_\varepsilon$ are only easier.

There can be 3 other possible 'moving' applications of $\rightarrowtail$ depending on the tree type of the node configuration in question. We will only analyze one case, as the others are symmetric. Suppose that we have

$$\rho = \langle p, (\triangledown, \bar{\rhd}), (a, d), H\rangle \rightarrowtail \tau = \langle p1, \alpha_1, (a_1, d_1), H_1\rangle$$

where $\rho \rightarrow_\triangledown \tau$. Let $\rho' = \langle p', (\triangledown, \bar{\rhd}), (a, d'), H'\rangle \prec_f \rho$. If ρ' is such that $\rho' \prec \tau$, the relation is trivially compatible. Otherwise, we shall prove that there is τ' such that $\rho' \rightarrowtail \tau'$ and $\tau' \prec \tau$. Condition (i) of $\rightarrow_\triangledown$ holds for ρ', because all the states present in ρ' are also in ρ (by definition of $\prec_f$) where the condition must hold. Then, we can apply the $\rightarrow_\triangledown$ transition to ρ' and obtain τ' of the form $\langle p'1, \alpha_1, (a_1, d_1'), H_1'\rangle$. Notice that we are taking α_1 and a_1 exactly as in τ, and that H_1' is completely determined by the $\rightarrow_\triangledown$ transition from H'. We only need to describe the value d_1' that will serve our purpose. As before, if $d_1 \in Im(f)$ we take $d_1' = f^{-1}(d_1)$ and check $\tau' \prec_f \tau$; and if $d_1 \notin Im(f)$ we take d_1' to be a new value not in $data(H')$ and check $\tau' \prec_{f'} \tau$ with $f' := f[d_1' \mapsto d_1]$. $\square$

We just showed that for *node configurations*, $(\mathsf{Conf}_\mathcal{N}, \rightarrowtail)$ is *rdc* w.r.t. $(\mathsf{Conf}_\mathcal{N}, \prec)$. We now lift this result to *tree configurations*, by considering that a tree configuration can be equivalently seen as an element from $(\mathsf{Conf}_\mathcal{N})^*$, and showing that the transition system $((\mathsf{Conf}_\mathcal{N})^*, \overset{t}{\rightarrowtail})$ is *rdc* w.r.t. the *embedding order* over $(\mathsf{Conf}_\mathcal{N}, \prec)$ that we define next.

Definition 3.6. *The* **embedding order** $(\mathcal{A}^*, \sqsubset)$ *over an order* $(\mathcal{A}, \leq)$ *is defined as follows.*

$(w_1 \cdots w_n) \sqsubset (v_1 \cdots v_m)$ *iff there exist* $1 \leq i_1 < i_2 < \cdots < i_n \leq m$ *such that* $w_j \leq v_{i_j}$ *for all* $j \in [1..n]$.

The lifting result is a standard argument, and can be stated in this general proposition, whose proof can be found in the Appendix.

Proposition 3.4. *Let* $\leq, \rightarrow_1 \subseteq \mathcal{A} \times \mathcal{A}$, $\sqsubset, \rightarrow_2 \subseteq \mathcal{A}^* \times \mathcal{A}^*$ *where* $\sqsubset$ *is the* embedding order *over* $(\mathcal{A}, \leq)$ *and* $\rightarrow_2$ *is such that if* $s \rightarrow_2 t$ *then:* s *and* t *are of the form* $s = \bar{u}\, a\, \bar{v}$, $t = \bar{u}\, \bar{b}\, \bar{v}$ *where* $\bar{b} = b_1 \cdots b_m$ *such that* $a \rightarrow_1 b_z$ *for every* $z \in [1..m]$. *Then,*

if $(\mathcal{A}, \leq)$ *is a* wqo *which is* rdc *with* $\rightarrow_1$,
then $(\mathcal{A}^*, \sqsubset)$ *is a* wqo *which is* rdc *with* $\rightarrow_2$.

We can apply this proposition by taking $\rightarrow_1$ as $\rightarrowtail$, $\leq$ as $\prec$, and taking that a $\mathsf{Conf}_\mathcal{T}$ configuration can be seen as an element of $(\mathsf{Conf}_\mathcal{N})^*$ by sorting the set by the lexicographic order on the first component (i.e., the node's position on the tree), and vice versa every element of $(\mathsf{Conf}_\mathcal{N})^*$ can be seen as an element from $\mathsf{Conf}_\mathcal{T}$. We instantiate $\rightarrow_2$ to be $\overset{t}{\rightarrowtail}$ as it verifies the conditions demanded for $\rightarrow_2$. As a result we have that $(\mathsf{Conf}_\mathcal{T}, \overset{t}{\rightarrowtail})$ is rdc w.r.t. $(\mathsf{Conf}_\mathcal{T}, \sqsubset)$ and the condition (1) of Proposition 3.3 is met.

As already mentioned, the transition $\overset{t}{\rightarrowtail}$ does not need to have infinite branching. This is just a consequence of the fact that the $\overset{t}{\rightarrowtail}$-image of any configuration has only a finite number of configurations up to isomorphism of the data values contained (remember that only equality between data values matters), and representatives for every class are effectively computable. We can then take $\overset{t}{\rightarrowtail}_2 \subset \overset{t}{\rightarrowtail}$ to have only one representative element for each class of equivalence and it then follows that the reachable classes of equivalence of $\overset{t}{\rightarrowtail}$ and $\overset{t}{\rightarrowtail}_2$ are the same. Hence, we have that condition (2) from Proposition 3.3 is also met. Finally, condition (3) holds as $\sqsubset$ is a wqo (by Proposition 3.4) that is computable. We can then apply Proposition 3.3 and it follows that the reachability and non-emptiness problems are decidable. Indeed, an $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ $\mathcal{M}$ is non-empty iff there exists an element of the finite basis of

$$\uparrow Succ^*(\{\langle \varepsilon, \alpha, (a, d_0), \{\langle q_I, d_0\rangle\}\rangle\})$$

—for any fixed d_0 and some $\alpha \in \{\triangledown, \bar{\triangledown}\} \times \{\rhd, \bar{\rhd}\}$ and $a \in \Sigma$— in which every node configuration has an empty set of threads.

4. DECIDABILITY OF XPATH

This section is mainly dedicated to the decidability of the satisfiability problem for a fragment of XPath with downward and rightward axes known as 'forward-XPath'. This is proved by a reduction to the $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ non-emptiness problem.

4.1 Definitions

We consider a fragment of the navigational part of XPath 1.0 with data equality and inequality. In particular this logic is here defined over *data trees*. However, an XML document may typically have not *one* data value per node, but a set of *attributes*, each carrying a data value. This is not a problem since every attribute of an XML element can be encoded as a child node in a data tree labeled by the attribute's name. Thus, all the decidability results hold also for XPath with attributes over XML documents.

Let us define a simplified syntax for this logic. XPath is a two-sorted language, with *path* expressions ($\alpha, \beta, \dots$) and *node* expressions ($\varphi, \psi, \dots$). We write $\mathsf{XPath}(\mathcal{O})$ to denote the data-aware fragment with the set of axes $\mathcal{O} \subseteq \{\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \leftarrow, {}^*\!\leftarrow, \uparrow, \uparrow^*\}$. It is defined by mutual recursion as follows,

$$\alpha, \beta ::= o \mid [\varphi] \mid \alpha\beta \mid \alpha \cup \beta \qquad o \in \mathcal{O} \cup \{\varepsilon\}$$

$$\varphi, \psi ::= a \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle\alpha\rangle \mid \alpha = \beta \mid \alpha \neq \beta$$

where $a \in \Sigma$, and Σ is a finite alphabet. A *formula* of $\mathsf{XPath}(\mathcal{O})$ is either a node expression or a path expression. We define the 'forward' set of axes as $\mathfrak{F} := \{\downarrow, \downarrow^*, \rightarrow, \rightarrow^*\}$, and consequently the fragment 'forward-XPath' as $\mathsf{XPath}(\mathfrak{F})$. We also refer by $\mathsf{XPath}^-(\mathfrak{F})$ to the fragment considered in [13] where data tests are of the restricted form $\varepsilon = \alpha$ or $\varepsilon \neq \alpha$.[2]

There have been efforts to extend XPath to have the full expressivity of MSO, e.g. by adding a least fix-point operator (cf. [3, Sect. 4.2]), but these logics generally lack clarity and simplicity. However, a form of recursion can be added by means of the Kleene star, which allows us to take the transitive closure of any path expression. Although in general this is not enough to already have MSO [4], it does give an intuitive language with counting ability. By $\mathsf{regXPath}(\mathcal{O})$ we refer to the enriched language where path expressions are extended by allowing the Kleene star on *any* path expression.

$$\alpha, \beta ::= o \mid [\varphi] \mid \alpha\beta \mid \alpha \cup \beta \mid \alpha^* \qquad o \in \mathcal{O} \cup \{\varepsilon\}$$

Let $\mathcal{T} = \langle P, \sigma\rangle$ be a data tree. The semantics of XPath is defined as the set of elements (in the case of node expressions) or pairs of elements (in the case of path expressions) selected by the expression. The data aware expressions are the cases $\alpha = \beta$ and $\alpha \neq \beta$.

$$\begin{aligned}
\llbracket\rightarrow\rrbracket^{\mathcal{T}} &:= \{(xi, x(i+1)) \mid x(i+1) \in P\}\\
\llbracket\downarrow\rrbracket^{\mathcal{T}} &:= \{(x, xi) \mid xi \in P\}\\
\llbracket\alpha^*\rrbracket^{\mathcal{T}} &:= \text{the reflexive transitive closure of } \llbracket\alpha\rrbracket^{\mathcal{T}}\\
\llbracket\varepsilon\rrbracket^{\mathcal{T}} &:= \{(x, x) \mid x \in P\}\\
\llbracket\alpha\beta\rrbracket^{\mathcal{T}} &:= \{(x, z) \mid \exists y.(x, y) \in \llbracket\alpha\rrbracket^{\mathcal{T}}, (y, z) \in \llbracket\beta\rrbracket^{\mathcal{T}}\}\\
\llbracket[\varphi]\rrbracket^{\mathcal{T}} &:= \{(x, x) \mid x \in \llbracket\varphi\rrbracket^{\mathcal{T}}\}\\
\llbracket\alpha \cup \beta\rrbracket^{\mathcal{T}} &:= \llbracket\alpha\rrbracket^{\mathcal{T}} \cup \llbracket\beta\rrbracket^{\mathcal{T}}\\
\llbracket\neg\varphi\rrbracket^{\mathcal{T}} &:= P \setminus \llbracket\varphi\rrbracket^{\mathcal{T}}\\
\llbracket\langle\alpha\rangle\rrbracket^{\mathcal{T}} &:= \{x \in P \mid \exists y.(x, y) \in \llbracket\alpha\rrbracket^{\mathcal{T}}\}\\
\llbracket\varphi \vee \psi\rrbracket^{\mathcal{T}} &:= \llbracket\varphi\rrbracket^{\mathcal{T}} \cup \llbracket\psi\rrbracket^{\mathcal{T}}\\
\llbracket\alpha = \beta\rrbracket^{\mathcal{T}} &:= \{x \in P \mid \exists y, z.(x, y) \in \llbracket\alpha\rrbracket^{\mathcal{T}},\\
&\qquad (x, z) \in \llbracket\beta\rrbracket^{\mathcal{T}}, \pi_2(\sigma(y)) = \pi_2(\sigma(z))\}\\
\llbracket\varphi \wedge \psi\rrbracket^{\mathcal{T}} &:= \llbracket\varphi\rrbracket^{\mathcal{T}} \cap \llbracket\psi\rrbracket^{\mathcal{T}}\\
\llbracket\alpha \neq \beta\rrbracket^{\mathcal{T}} &:= \{x \in P \mid \exists y, z.(x, y) \in \llbracket\alpha\rrbracket^{\mathcal{T}},\\
&\qquad (x, z) \in \llbracket\beta\rrbracket^{\mathcal{T}}, \pi_2(\sigma(y)) \neq \pi_2(\sigma(z))\}\\
\llbracket a\rrbracket^{\mathcal{T}} &:= \{x \in P \mid \pi_1(\sigma(x)) = a\}
\end{aligned}$$

For instance, in the model of Fig. 1,

$$\llbracket\langle\downarrow^*[b \wedge \downarrow[b] \neq \downarrow[b]]\rangle\rrbracket^{\mathcal{T}} = \{\epsilon, 1, 12\}.$$

We define $\mathsf{sub}(\varphi)$ to denote the set of all subformulæ of φ, $\mathsf{psub}(\varphi) := \{\alpha \mid \alpha \in \mathsf{sub}(\varphi), \alpha \text{ is a path expression}\}$, and $\mathsf{nsub}(\varphi) := \{\psi \mid \alpha \in \mathsf{sub}(\varphi), \psi \text{ is a node expression}\}$.

Primary key. It is worth noting that $\mathsf{XPath}(\mathfrak{F})$ —contrary to $\mathsf{XPath}^-(\mathfrak{F})$— can express unary *primary key* constraints. That is, whether for some symbol a, all the a-elements in the tree have different data values.

LEMMA 4.1. *For every $a \in \Sigma$ let $pk(a)$ be the property over a tree $\mathcal{T} = \langle P, \sigma\rangle$: "For every two different positions $p, p' \in P$ of the tree, if $\pi_1(p) = \pi_1(p') = a$, then $\pi_2(p) \neq \pi_2(p')$". Then, $pk(a)$ is expressible in* XPath *for any a.*

PROOF. It is easy to see that the *negation* of this property can be tested by first *guessing* the closest common ancestor of two different a-elements with equal datum in the underlying 'first child'-'next sibling' binary tree coding. At this node, we verify the presence of two a-nodes with equal datum, one accessible with a "$\downarrow^*$" relation and the other with a compound "$\rightarrow^+\downarrow^*$" relation (hence the nodes are different). The expressibility of the property then follows from the logic being closed under negation. The reader can check that the following formula expresses the property, where '$\downarrow^+$' = '$\downarrow\downarrow^*$' and '$\rightarrow^+$' = '$\rightarrow\rightarrow^*$'.

$$pk(a) \equiv \neg\langle\downarrow^*[\,\varepsilon[a] = \downarrow^+[a] \vee \downarrow^*[a] = \rightarrow^+\downarrow^*[a]\,]\rangle \quad \square$$

4.2 Reduction to ATRA non-emptiness

In this section we show how satisfiability of forward-XPath can be decided with the help of the automata model introduced in Section 3. First let us fix some nomenclature.

DEFINITION 4.1. *We say that a class of automata $\mathcal{S}$ **captures** a logic $\mathcal{L}$ iff there exists a translation $t : \mathcal{L} \rightarrow \mathcal{S}$ such that for every $\varphi \in \mathcal{L}$ and data tree $\mathcal{T}$, we have that $\mathcal{T}$ verifies φ if and only if $t(\varphi)$ has an accepting run over $\mathcal{T}$.*

[13] shows that ATRA captures the fragment '$\mathsf{XPath}^-(\mathfrak{F})$'. It is immediate to see that ATRA can easily capture the Kleene star operator on any path formula, obtaining decidability of $\mathsf{regXPath}^-(\mathfrak{F})$. However, these decidability results cannot be generalized to the full unrestricted forward fragment $\mathsf{XPath}(\mathfrak{F})$ as ATRA is not powerful enough to capture the expressivity of the logic. It cannot express, for instance, that there are two different leaves with the same data value. Unfortunately, the model ATRA(guess, spread) introduced in this article can neither capture $\mathsf{XPath}(\mathfrak{F})$. Concretely, data tests of the form $\neg(\alpha = \beta)$ are impossible to perform for ATRA(guess, spread) as this would require —in some sense— the ability to guess two disjoint sets of data values S_1, S_2

[2] [13] refers to $\mathsf{XPath}^-(\mathfrak{F})$ as 'forward XPath'. Here, 'forward XPath' is the unrestricted fragment $\mathsf{XPath}(\mathfrak{F})$.

such that all α-paths lead to a data value of S_1, and all β-paths lead to a data value of S_2. Still, in the sequel we show that there exists a reduction from the satisfiability of $\mathsf{regXPath}(\mathfrak{F})$ to the emptiness of $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$. This result settles an open question regarding the decidability of the satisfiability problem for the forward-XPath fragment: $\mathsf{XPath}(\mathfrak{F})$. The main results that will be shown in Section 4.3 are the following.

Theorem 4.1. *Satisfiability of* $\mathsf{regXPath}(\mathfrak{F})$ *in the presence of DTDs and unary primary key constraints is decidable, non primitive recursive.*

And hence the next corollary follows from the logic being closed under boolean operations.

Corollary 4.1. *The query containment and the query equivalence problems are decidable for* $\mathsf{XPath}(\mathfrak{F})$.

Moreover, these decidability results hold for $\mathsf{regXPath}(\mathfrak{F})$ and even for two extensions:

- a navigational extension with *upward* axes (in Section 4.4), and
- a generalization of the data tests that can be performed (in Section 4.5).

4.3 Allowing arbitrary data tests

This section is devoted to the proof of the following statement.

Proposition 4.1. *For every* $\eta \in \mathsf{regXPath}(\mathfrak{F})$ *there exists an effectively computable* $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ *automaton* $\mathcal{M}$ *such that* $\mathcal{M}$ *is non-empty iff* η *is satisfiable.*

The proof can be sketched as follows:

- We define a *strategy* of evaluation consisting of a restriction of the transition $\overset{t}{\rightarrowtail}$ of $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ that is referred to as $\underset{\cdot}{\overset{t}{\rightarrowtail}} \subseteq \overset{t}{\rightarrowtail}$. This strategy verifies that there exists an accepting run under $\overset{t}{\rightarrowtail}$ iff there exists an accepting run under $\underset{\cdot}{\overset{t}{\rightarrowtail}}$.
- We give a translation from forward XPath formulæ to $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automata such that (1) any tree accepted by the automaton $\mathcal{M}$ with the evaluation strategy $\underset{\cdot}{\overset{t}{\rightarrowtail}}$ verifies the XPath formula η, and (2) any tree verified by the formula η is accepted by the automaton $\mathcal{M}$.

Intuitively, $\underset{\cdot}{\overset{t}{\rightarrowtail}}$ is the restriction of $\overset{t}{\rightarrowtail}$ to a finitely branching transition system, where each data value introduced non deterministically (either from a guess or from a $\to_\triangledown$ or $\to_\triangleright$ transition) verifies that either it already existed in the current node configuration, or it has not appeared so far along the whole execution of the automaton. Note that with this semantics the automaton accepts strictly *less* data trees. However, this can be done preserving the existence of data trees with accepting runs since from an accepting $\overset{t}{\rightarrowtail}$ sequence one can construct a similar accepting $\underset{\cdot}{\overset{t}{\rightarrowtail}}$ sequence. Indeed, for any data tree $\mathcal{T}$ accepted under the $\overset{t}{\rightarrowtail}$ semantics, there is another data tree $\mathcal{T}'$, that only differs in $\mathcal{T}$ in the data values of some nodes, which is accepted both under the $\underset{\cdot}{\overset{t}{\rightarrowtail}}$ semantics and the $\overset{t}{\rightarrowtail}$ semantics.

Definition 4.2. *Let us fix* $\theta : \mathbb{N}^* \times \mathbb{N} \to \mathbb{D}$ *an injective map. Consider the restriction where* $\rho = \langle p, \ldots \rangle \to_\triangledown \tau$ *applies only if*

1. τ *introduces a data value already in* $\mathsf{data}(\rho)$*, or*
2. *if it introduces the value* $\theta(p1, 1)$.

A similar restriction applies for $\rho = \langle pi, \ldots \rangle \to_\triangleright \tau$ *and* $\theta(p(i+1), 1)$. *Finally,* $\rho = \langle p, \ldots \rangle \to_\varepsilon \tau$ *applies only if either*

1. *a non-*guess *transition is performed;*
2. *a* guess *transition is performed and a data value already in* $\mathsf{data}(\rho)$ *is guessed; or*
3. *a* guess *transition is performed and the guessed data value is* $\theta(p, i_{min})$*, where*

$$i_{min} = min\{i \mid \theta(p, i) \notin \mathsf{data}(\rho)\}.$$

We note this restriction of $\overset{t}{\rightarrowtail}$ *by* $\underset{\cdot}{\overset{t}{\rightarrowtail}}$.

The following lemma follows from the definition above(see the Appendix for more details).

Lemma 4.2. *Let* $\mathcal{M}$ *be an* $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$. $\mathcal{M}$ *has an accepting run under* $\overset{t}{\rightarrowtail}$ *iff it has an accepting run under* $\underset{\cdot}{\overset{t}{\rightarrowtail}}$.

Based on the semantics of $\underset{\cdot}{\overset{t}{\rightarrowtail}}$, we define a translation from $\mathsf{regXPath}(\mathfrak{F})$ formulæ to $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automata. Let η be a $\mathsf{regXPath}(\mathfrak{F})$ formula and let $\mathcal{M}$ be the corresponding $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automaton defined by the translation. We show that (i) if a data tree $\mathcal{T}$ is accepted by $\mathcal{M}$ under the $\underset{\cdot}{\overset{t}{\rightarrowtail}}$ strategy, then $\mathcal{T}$ verifies η, and (ii) if a data tree $\mathcal{T}$ verifies η, then $\mathcal{T}$ is accepted by $\mathcal{M}$ (under $\overset{t}{\rightarrowtail}$). The emptiness problem for $\mathcal{M}$ under $\overset{t}{\rightarrowtail}$ and $\underset{\cdot}{\overset{t}{\rightarrowtail}}$ are equivalent as already discussed, and thus Proposition 4.1 follows.

The translation

Let η be a $\mathsf{regXPath}(\mathfrak{F})$ node expression in negated normal form (nnf for short). For succinctness and simplicity of the translation, we assume that η is in a normal form such that the $\downarrow$-axis is interpreted as the *leftmost* child. To obtain this normal form, it suffices to replace every appearance of '$\downarrow$' by '$\downarrow\to^*$'. For every path expression $\alpha \in \mathsf{psub}(\eta)$, consider a deterministic complete finite automaton $\mathcal{H}_\alpha$ over the alphabet $\Sigma_\eta = \{\varphi \mid \varphi \in \mathsf{nsub}(\eta)\} \cup \{\downarrow, \to\}$ which corresponds to that regular expression. We assume the following names of its components: $\mathcal{H}_\alpha = \langle \Sigma_\eta, \delta_\alpha, Q_\alpha, 0, F_\alpha \rangle$, with $Q_\alpha \subset \mathbb{N}$ the finite set of states and $0 \in Q_\alpha$ the initial state. We next show how to translate η into an $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automaton $\mathcal{M}$. For the sake of readability we define the transitions as positive boolean combinations of $\vee$ and $\wedge$ over the set of basic tests and states. Any of these —take for instance $\delta(q) = (\mathsf{set}(q_1) \wedge \triangledown q_2) \vee (q_3 \wedge \bar{\mathsf{a}})$— can be rewritten into an equivalent ATRA with at most one boolean connector per transition (as in Definition 3.1) in polynomial time. The most important cases are those relative to the following data tests:

1. $\alpha = \beta$ 2. $\alpha \neq \beta$ 3. $\neg(\alpha = \beta)$ 4. $\neg(\alpha \neq \beta)$

We define the $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automaton

$$\mathcal{M} := \langle \Sigma, Q, \llparenthesis\varphi\rrparenthesis, \delta \rangle$$

with

$$\begin{aligned} Q := \{&\llparenthesis\varphi\rrparenthesis, \llparenthesis\alpha\rrparenthesis^{\circledast}_{\mathcal{C},i}, \llparenthesis\alpha\rrparenthesis^{\circledast}_{\mathsf{F}}, \llparenthesis\alpha,\beta\rrparenthesis^{\circledast}_{\mathcal{C},i,\mathcal{E},j} \mid \varphi \in \mathsf{nsub}^{\neg}(\eta), \\ &\alpha, \beta \in \mathsf{psub}^{\neg}(\eta), \circledast \in \{=, \neq, \neg=, \neg\neq\}, \\ &i \in Q_\alpha, \mathcal{C} \subseteq Q_\alpha, j \in Q_\beta, \mathcal{E} \subseteq Q_\beta\} \end{aligned}$$

where $\mathsf{op}^{\neg}$ is the smallest superset of op closed under negation under nnf, i.e., if $\varphi \in \mathsf{op}^{\neg}(\eta)$ then $\mathsf{nnf}(\neg\varphi) \in \mathsf{op}^{\neg}(\eta)$. The sets $\mathcal{C}, \mathcal{E}$ are not essential to understand the general construction, and they have as only purpose to disallow non-moving loops in the definition of δ. As an example we first take care of the boolean connectors and the simplest tests.

$$\begin{aligned} \delta(\llparenthesis\mathsf{a}\rrparenthesis) &:= \mathsf{a} & \delta(\llparenthesis\varphi \vee \psi\rrparenthesis) &:= \llparenthesis\varphi\rrparenthesis \vee \llparenthesis\psi\rrparenthesis \\ \delta(\llparenthesis\neg\mathsf{a}\rrparenthesis) &:= \bar{\mathsf{a}} & \delta(\llparenthesis\varphi \wedge \psi\rrparenthesis) &:= \llparenthesis\varphi\rrparenthesis \wedge \llparenthesis\psi\rrparenthesis \end{aligned}$$

The tests $\langle\alpha\rangle$ and $\neg\langle\alpha\rangle$ are coded in a standard way, see[13] for more details. Here we focus on the data-aware cases. Using the guess operator, we can easily define the cases corresponding to the data test cases 1 and 2 as follows. Here, $\llparenthesis\alpha\rrparenthesis_{\mathsf{F}}$ holds at the endpoint of a path matching α.

$$\begin{aligned} &\delta(\llparenthesis\alpha = \beta\rrparenthesis) := \mathsf{guess}(\llparenthesis\alpha,\beta\rrparenthesis^{=}) \\ &\delta(\llparenthesis\alpha,\beta\rrparenthesis^{=}) := \llparenthesis\alpha\rrparenthesis^{=}_{\emptyset,0} \wedge \llparenthesis\beta\rrparenthesis^{=}_{\emptyset,0} \qquad \delta(\llparenthesis\alpha\rrparenthesis^{=}_{\mathsf{F}}) := \mathsf{eq} \\ &\delta(\llparenthesis\alpha \neq \beta\rrparenthesis) := \mathsf{guess}(\llparenthesis\alpha,\beta\rrparenthesis^{\neq}) \\ &\delta(\llparenthesis\alpha,\beta\rrparenthesis^{\neq}) := \llparenthesis\alpha\rrparenthesis^{=}_{\emptyset,0} \wedge \llparenthesis\beta\rrparenthesis^{\neq}_{\emptyset,0} \qquad \delta(\llparenthesis\alpha\rrparenthesis^{\neq}_{\mathsf{F}}) := \overline{\mathsf{eq}} \end{aligned}$$

We define the transitions associated to each $\mathcal{H}_\alpha$, for $i \in Q_\alpha, \mathcal{C} \subseteq Q_\alpha, \circledast \in \{=, \neq\}$.

$$\begin{aligned} \delta(\llparenthesis\alpha\rrparenthesis^{\circledast}_{\mathcal{C},i}) := &\bigvee_{\substack{\varphi \in \mathsf{nsub}(\alpha), \\ i' := \delta_\alpha(\varphi,i),\ i' \notin \mathcal{C}}} (\llparenthesis\varphi\rrparenthesis \wedge \llparenthesis\alpha\rrparenthesis^{\circledast}_{\mathcal{C}\cup\{i'\},i'}) \\ &\vee\ \triangledown\llparenthesis\alpha\rrparenthesis^{\circledast}_{\emptyset,\delta_\alpha(\downarrow,i)} \vee \triangleright\llparenthesis\alpha\rrparenthesis^{\circledast}_{\emptyset,\delta_\alpha(\rightarrow,i)} \vee \bigvee_{i \in F_\alpha} \llparenthesis\alpha\rrparenthesis^{\circledast}_{\mathsf{F}} \end{aligned}$$

The test case 4 involves also an *existential* quantification over data values. In fact, $\neg(\alpha \neq \beta)$ means that either (1) there are no nodes reachable by α, or (2) there are no nodes reachable by β, or (3) there *exists* a data value d such that both (a) all elements reachable by α have datum d, and (b) all elements reachable by β have datum d.

$$\begin{aligned} &\delta(\llparenthesis\neg\alpha \neq \beta\rrparenthesis) := \llparenthesis\neg\langle\alpha\rangle\rrparenthesis \vee \llparenthesis\neg\langle\beta\rangle\rrparenthesis \vee \mathsf{guess}(\llparenthesis\alpha,\beta\rrparenthesis^{\neg\neq}) \\ &\delta(\llparenthesis\alpha,\beta\rrparenthesis^{\neg\neq}) := \llparenthesis\alpha\rrparenthesis^{\neg\neq}_{\emptyset,0} \wedge \llparenthesis\beta\rrparenthesis^{\neg\neq}_{\emptyset,0} \\ &\delta(\llparenthesis\alpha\rrparenthesis^{\neg\neq}_{\mathsf{F}}) := \mathsf{eq} \qquad \delta(\llparenthesis\alpha\rrparenthesis^{\neg=}_{\mathsf{F}}) := \overline{\mathsf{eq}} \end{aligned}$$

$$\begin{aligned} \delta(\llparenthesis\alpha\rrparenthesis^{\neg\circledast}_{\mathcal{C},i}) := &\bigwedge_{\substack{\varphi \in \mathsf{nsub}(\alpha), \\ i' := \delta_\alpha(\varphi,i),\ i' \notin \mathcal{C}}} (\llparenthesis\bar{\varphi}\rrparenthesis \vee \llparenthesis\alpha\rrparenthesis^{\neg\circledast}_{\mathcal{C}\cup\{i'\},i'}) \\ &\wedge (\bar{\triangledown}? \vee \triangledown\llparenthesis\alpha\rrparenthesis^{\neg\circledast}_{\emptyset,\delta_\alpha(\downarrow,i)}) \wedge (\bar{\triangleright}? \vee \triangleright\llparenthesis\alpha\rrparenthesis^{\neg\circledast}_{\emptyset,\delta_\alpha(\rightarrow,i)}) \\ &\wedge \bigwedge_{i \in F_\alpha} \llparenthesis\alpha\rrparenthesis^{\neg\circledast}_{\mathsf{F}} \quad \text{where } \bar{\varphi} \text{ stands for } \mathsf{nnf}(\neg\varphi). \end{aligned}$$

The difficult part is the translation of the data test case 3. The main reason for this difficulty is the fact that ATRA automata do not have the expressivity to make these kinds of tests. An expression $\neg(\alpha = \beta)$ forces the set of data values reachable by an α-path and the set of those reachable by a β-path to be disjoint. We show that nonetheless the automaton can test for a condition that is sat-equivalent to $\neg(\alpha = \beta)$. Suppose first that $\eta = \neg(\downarrow \alpha = \rightarrow \beta)$ is to be checked for satisfiability. One obvious answer would be to test separately α and β. If both tests succeed, we can then build a model satisfying η out of the two witnessing trees by making sure they have disjoint sets of values. Otherwise, η is clearly unsatisfiable. Suppose now that we have $\eta = \varphi \wedge \neg(\downarrow \alpha = \rightarrow \beta)$, where φ is any formula with no data tests of type 3. One could build the automaton for φ and then ask for "$\mathsf{spread}(\llparenthesis \downarrow \alpha\rrparenthesis^{\neg=}_{0} \vee \llparenthesis \rightarrow \beta\rrparenthesis^{\neg=}_{0})$" in the automaton. This corresponds to the property "*for every data value d taken into account by the automaton (as a result of the translation of φ), either all elements reachable by α do not have datum d, or all elements reachable by β do not have datum d*". If φ contains a $\alpha' = \beta'$ formula, this translates to a *guessing* of a witnessing data value d. Then the use of spread takes care of this particular data value, and indeed of all other data values that were guessed to satisfy similar demands. In other words, it is *not* because of d that $\neg(\downarrow \alpha = \rightarrow \beta)$ will be falsified. But then, the $\overset{t}{\rightarrowtail}$ semantics ensures that no pair of nodes accessible by α and β share the same datum. This is the main idea we encode next. Here, $\mathsf{spread}(q) := \bigwedge_{q' \in Q} \mathsf{spread}(q', q)$, and we define $\delta(\llparenthesis\neg(\alpha = \beta)\rrparenthesis) := \llparenthesis\alpha,\beta\rrparenthesis^{\neg=}_{\emptyset,0,\emptyset,0}$. Given $\neg(\alpha = \beta)$, the automaton systematically looks for the closest common ancestor of every pair (x, y) of nodes accessible by α and β respectively, and tests, for every data value d in the node configuration, that either (1) all data values accessible by the remaining path of α are different from d, or (2) all data values accessible by the remaining path of β are different from d.

$$\begin{aligned} \delta(\llparenthesis\alpha,\beta\rrparenthesis^{\neg=}_{\mathcal{C}_1,i,\mathcal{C}_2,j}) := &\ \mathsf{spread}\big(\llparenthesis\alpha\rrparenthesis^{\neg=}_{\emptyset,i} \vee \llparenthesis\beta\rrparenthesis^{\neg=}_{\emptyset,j}\big) \\ &\wedge \triangledown\llparenthesis\alpha,\beta\rrparenthesis^{\neg=}_{\emptyset,\delta_\alpha(\downarrow,i),\emptyset,\delta_\beta(\downarrow,j)} \\ &\wedge \triangleright\llparenthesis\alpha,\beta\rrparenthesis^{\neg=}_{\emptyset,\delta_\alpha(\rightarrow,i),\emptyset,\delta_\beta(\rightarrow,j)} \\ &\wedge \bigwedge_{i \in F_\alpha} \llparenthesis\beta\rrparenthesis^{\neg=}_{\emptyset,j} \wedge \bigwedge_{j \in F_\alpha} \llparenthesis\alpha\rrparenthesis^{\neg=}_{\emptyset,i} \\ &\wedge \bigwedge_{\substack{\varphi \in \mathsf{nsub}(\alpha), \\ i' := \delta_\alpha(\varphi,i),\ i' \notin \mathcal{C}_1}} (\llparenthesis\bar{\varphi}\rrparenthesis \vee \llparenthesis\alpha,\beta\rrparenthesis^{\neg=}_{\mathcal{C}_1\cup\{i'\},i',\mathcal{C}_2,j}) \\ &\wedge \bigwedge_{\substack{\varphi \in \mathsf{nsub}(\beta), \\ j' := \delta_\beta(\varphi,j),\ j' \notin \mathcal{C}_2}} (\llparenthesis\bar{\varphi}\rrparenthesis \vee \llparenthesis\alpha,\beta\rrparenthesis^{\neg=}_{\mathcal{C}_1,i,\mathcal{C}_2\cup\{j'\},j'}) \end{aligned}$$

The following lemmas then follow from the discussion above.

LEMMA 4.3. *For any data tree $\mathcal{T}$, if $\mathcal{T}$ verifies η, then $\mathcal{M}$ accepts $\mathcal{T}$ under the $\overset{t}{\rightarrowtail}$ semantics.*

LEMMA 4.4. *For any data tree $\mathcal{T}$, if $\mathcal{M}$ accepts $\mathcal{T}$ under the $\overset{t}{\underset{\cdot}{\rightarrowtail}}$ semantics, then $\mathcal{T}$ verifies η.*

Lemmas 4.3 and 4.4 together with Lemma 4.2 conclude the proof of Proposition 4.1. We then have that Theorem 4.1 holds.

PROOF OF THEOREM 4.1. By Proposition 4.2, satisfiability of $\mathsf{regXPath}(\mathfrak{F})$ is reducible to the nonemptiness problem for $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$. On the other hand, we remark that $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automata can code any regular tree language —in particular a DTD, the core of XML Schema, or Relax NG— and are closed under intersection by Proposition 3.2. Also, the logic can express any unary primary key constraint as stated in Lemma 4.1. Hence, by Theorem 3.1 the decidability follows.

It is known that even much simpler fragments of XPath have non primitive recursive complexity [9]. □

4.4 Allowing upward axes

Here we explore one possible decidable extension to the logic $\mathsf{regXPath}(\mathfrak{F})$, whose decidability can be reduced to that of $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$.

Consider the data test expressions of the types

$$\neg(\alpha_{\mathfrak{b}} = \beta_{\mathfrak{f}}) \qquad \text{and} \qquad \neg(\alpha_{\mathfrak{b}} \neq \beta_{\mathfrak{f}})$$

where $\beta_{\mathfrak{f}} \in \mathsf{regXPath}(\mathfrak{F})$ and $\alpha_{\mathfrak{b}} \in \mathsf{regXPath}(\mathfrak{B})$, with $\mathfrak{B} := \{\uparrow, \uparrow^*, \leftarrow, {}^*\!\!\leftarrow\}$. We can decide the satisfaction of these kinds of expressions by means of the $\mathsf{spread}(\cdot, \cdot)$, using carefully its first parameter to select the desired threads from which to collect the data values we are interested in. Intuitively, along the run we throw threads that save current data value and try out all possible ways to verify $\alpha_{\mathfrak{b}}^r \in \mathsf{regXPath}(\mathfrak{F})$, where $\cdot^r$ stands for the *reverse* of the regular expression. Let the automaton arrive at a configuration $\langle \llparenthesis \alpha_{\mathfrak{b}} \rrparenthesis, d \rangle$ whenever $\alpha_{\mathfrak{b}}^r$ is verified. This signals that there is a backwards path from the current node in the relation $\alpha_{\mathfrak{b}}$ that arrives at a node with data value d. Hence, at any given position, the instruction $\mathsf{spread}(\llparenthesis \alpha_{\mathfrak{b}} \rrparenthesis, \llparenthesis \alpha_{\mathfrak{f}} \rrparenthesis^{\neg\circledast})$ translates correctly the expression $\neg(\alpha_{\mathfrak{b}} \circledast \beta_{\mathfrak{f}})$. Furthermore, $\alpha_{\mathfrak{b}}$ need not be necessarily in $\mathsf{regXPath}(\mathfrak{B})$, as its intermediate node tests can be formulæ from $\mathsf{regXPath}(\mathfrak{F})$. More formally, let $\mathsf{regXPath}^{\mathfrak{B}}(\mathfrak{F})$ be the fragment of $\mathsf{regXPath}(\mathfrak{F} \cup \mathfrak{B})$ defined by the grammar

$$\begin{aligned}\varphi, \psi ::= {} & \neg\mathsf{a} \mid \mathsf{a} \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle \alpha_{\mathfrak{f}} \rangle \mid \langle \alpha_{\mathfrak{b}} \rangle \mid \\ & \alpha_{\mathfrak{f}} \circledast \beta_{\mathfrak{f}} \mid \neg(\alpha_{\mathfrak{f}} \circledast \beta_{\mathfrak{f}}) \mid \neg(\alpha_{\mathfrak{b}} = \beta_{\mathfrak{f}}) \mid \neg(\alpha_{\mathfrak{b}} \neq \beta_{\mathfrak{f}})\end{aligned}$$

with $\circledast \in \{=, \neq\}, \mathsf{a} \in \Sigma$, and

$$\begin{aligned}\alpha_{\mathfrak{f}}, \beta_{\mathfrak{f}} &::= [\varphi] \mid \alpha_{\mathfrak{f}}\beta_{\mathfrak{f}} \mid o\,\alpha_{\mathfrak{f}} \mid (\alpha_{\mathfrak{f}})^* \qquad o \in \{\downarrow, \rightarrow, \varepsilon\}, \\ \alpha_{\mathfrak{b}}, \beta_{\mathfrak{b}} &::= [\varphi] \mid \alpha_{\mathfrak{b}}\beta_{\mathfrak{b}} \mid o\,\alpha_{\mathfrak{b}} \mid (\alpha_{\mathfrak{b}})^* \qquad o \in \{\uparrow, \leftarrow, \varepsilon\}.\end{aligned}$$

We must note that $\mathsf{regXPath}^{\mathfrak{B}}(\mathfrak{F})$ contains the full data-unaware fragment (i.e., with no data tests) of $\mathsf{regXPath}(\mathfrak{B})$, and that it is *not* closed under negation. In fact, were it closed under negation, its satisfiability would be undecidable. As mentioned, we can decide the satisfiability problem for this fragment.

THEOREM 4.2. *Satisfiability for $\mathsf{regXPath}^{\mathfrak{B}}(\mathfrak{F})$ under primary key constraints and DTDs is decidable.*

4.5 Allowing stronger data tests

Consider the property "*there are three descendant nodes labeled a, b and c with the same data value*". That is, there exists some data value d such that there are three nodes accessible by $\downarrow^* [a]$, $\downarrow^* [b]$ and $\downarrow^* [c]$ respectively, all carrying the datum d. Let us denote the fact that they have the same or different datum by introducing the symbols '$\sim$' and '$\nsim$', and appending it at the end of the path. Then in this case we write that the elements must satisfy $\downarrow^* [\mathsf{a}]{\sim}$, $\downarrow^* [\mathsf{b}]{\sim}$, and $\downarrow^* [\mathsf{c}]{\sim}$. We then introduce the node expression $\{\!\!\{\alpha_1 s_1, \ldots, \alpha_n s_n\}\!\!\}$ where α_i is a path expression and $s_i \in \{\sim, \nsim\}$ for all $i \in [1..n]$. Semantically, it is a node expression that denotes all the tree positions p from which we can access n nodes $p_1, \ldots, p_n$ such that there exists $d \in \mathbb{D}$ where for all $i \in [1..n]$ the following holds: $(p, p_i) \in [\![\alpha_i]\!]$; if $s_i = \sim$ then $\pi_2(\sigma(p_i)) = d$; and if $s_i = \nsim$ then $\pi_2(\sigma(p_i)) \neq d$. Note that now we can express $\alpha = \beta$ as $\{\!\!\{\alpha{\sim}, \beta{\sim}\}\!\!\}$ and $\alpha \neq \beta$ as $\{\!\!\{\alpha{\sim}, \beta{\nsim}\}\!\!\}$. Let us call $\mathsf{regXPath}^{+}(\mathfrak{F})$ to $\mathsf{regXPath}(\mathfrak{F})$ extended with the construction just explained. This is a more expressive formalism since the first mentioned property —or, to give another example, $\{\!\!\{\downarrow^* [\mathsf{a}]{\sim}, \downarrow^* [\mathsf{b}]{\sim}, \downarrow^* [\mathsf{a}]{\nsim}, \downarrow^* [\mathsf{b}]{\nsim}\}\!\!\}$— is not expressible in $\mathsf{regXPath}(\mathfrak{F})$.

We argue that satisfiability for this extension can be decided in the same way as for $\mathsf{regXPath}(\mathfrak{F})$. It is straightforward to see that *positive* appearances can easily be translated with the help of the guess operator. On the other hand, for negative appearances, like $\neg\{\!\!\{\alpha_1 s_1, \ldots, \alpha_n s_n\}\!\!\}$, we proceed in the same way as we did for $\mathsf{regXPath}(\mathfrak{F})$. The only difference being that in this case the automaton will simulate the simultaneous evaluation of the n expressions and calculate all possible configurations of the closest common ancestors of the endpoints, performing a spread at each of these intermediate points.

THEOREM 4.3. *Satisfiability of $\mathsf{regXPath}^{+}(\mathfrak{F})$ under primary key constraints and DTDs is decidable.*

5. CONCLUDING REMARKS

We presented a simplified framework to work with 1-way alternating register automata on data trees, enabling the possibility to easily show decidability of new operators by proving that they preserve the downward compatibility of a well-structured transition system. It would be interesting to hence explore more decidable extensions, to study the expressiveness limits of decidable logics and automata for data trees.

Also, this work argues in favor of exploring models that although they might be not closed under all boolean operations, can serve to show decidability of logics closed under negation —such as forward-XPath— or expressive natural extensions of existing logics.

Acknowledgment. We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

6. REFERENCES

[1] Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2008.

[2] Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. In *PODS*, pages 10–19. ACM Press, 2006.

[3] Balder ten Cate. The expressivity of XPath with transitive closure. In *PODS*, pages 328–337. ACM Press, 2006.

[4] Balder ten Cate and Luc Segoufin. XPath, transitive closure logic, and nested tree walking automata. In *PODS*, pages 251–260. ACM Press, 2008.

[5] James Clark and Steve DeRose. XML path language (XPath). Website, November 1999. W3C Recommendation. `http://www.w3.org/TR/xpath`.

[6] Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. In *LICS*, pages 17–26. IEEE Computer Society Press, 2006.

[7] Stéphane Demri, Ranko Lazić, and David Nowak. On the freeze quantifier in constraint LTL: Decidability and complexity. In *TIME*, pages 113–121. IEEE Computer Society Press, 2005.

[8] Diego Figueira. Satisfiability of downward XPath with data equality tests. In *PODS*, pages 197–206. ACM Press, 2009.

[9] Diego Figueira and Luc Segoufin. Future-looking logics on data words and trees. In *MFCS*, volume 5734 of *LNCS*, pages 331–343. Springer, 2009.

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

[11] Floris Geerts and Wenfei Fan. Satisfiability of XPath queries with sibling axes. In *DBPL*, volume 3774 of *LNCS*, pages 122–137. Springer, 2005.

[12] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.

[13] Marcin Jurdziński and Ranko Lazić. Alternating automata on data trees and XPath satisfiability. *CoRR*, abs/0805.0330, 2008.

[14] Maarten Marx. XPath with conditional axis relations. In *EDBT*, volume 2992 of *LNCS*, pages 477–494. Springer, 2004.

APPENDIX

A. DETAILED DEFINITIONS

$$\begin{aligned}
\mathsf{nnf}(\varphi \wedge \psi) &:= \mathsf{nnf}(\varphi) \wedge \mathsf{nnf}(\psi)\\
\mathsf{nnf}(\varphi \vee \psi) &:= \mathsf{nnf}(\varphi) \vee \mathsf{nnf}(\psi)\\
\mathsf{nnf}(\neg(\varphi \wedge \psi)) &:= \mathsf{nnf}(\neg\varphi) \vee \mathsf{nnf}(\neg\psi)\\
\mathsf{nnf}(\neg(\varphi \vee \psi)) &:= \mathsf{nnf}(\neg\varphi) \wedge \mathsf{nnf}(\neg\psi)\\
\mathsf{nnf}(o\,\alpha) &:= o\,\mathsf{nnf}(\alpha) \quad o \in \mathfrak{F}\\
\mathsf{nnf}([\varphi]\alpha) &:= [\mathsf{nnf}(\varphi)]\mathsf{nnf}(\alpha)\\
\mathsf{nnf}(\alpha \circledast \beta) &:= \mathsf{nnf}(\alpha) \circledast \mathsf{nnf}(\beta)\\
\mathsf{nnf}(\neg\alpha \circledast \beta) &:= \neg\mathsf{nnf}(\alpha) \circledast \mathsf{nnf}(\beta)\\
\mathsf{nnf}(\mathsf{a}) &:= \mathsf{a}\\
\mathsf{nnf}(\neg\mathsf{a}) &:= \neg\mathsf{a}\\
\mathsf{nnf}(\neg\neg\varphi) &:= \mathsf{nnf}(\varphi)\\
\mathsf{nnf}(\langle\alpha\rangle) &:= \langle\mathsf{nnf}(\alpha)\rangle
\end{aligned}$$

$$\begin{aligned}
\mathsf{sub}(\varphi \wedge \psi) &:= \{\varphi \wedge \psi\} \cup \mathsf{sub}(\varphi) \cup \mathsf{sub}(\psi)\\
\mathsf{sub}(\neg\varphi) &:= \{\neg\varphi\} \cup \mathsf{sub}(\varphi)\\
\mathsf{sub}(\varphi \vee \psi) &:= \{\varphi \vee \psi\} \cup \mathsf{sub}(\varphi) \cup \mathsf{sub}(\psi)\\
\mathsf{sub}(\mathsf{a}) &:= \{\mathsf{a}\}\\
\mathsf{sub}([\varphi]\alpha) &:= \{[\varphi]\alpha\} \cup \mathsf{sub}(\varphi) \cup \mathsf{sub}(\alpha)\\
\mathsf{sub}(o\,\alpha) &:= \{o\,\alpha\} \cup \mathsf{sub}(\alpha)\\
\mathsf{sub}(\alpha \circledast \beta) &:= \{\alpha \circledast \beta\} \cup \mathsf{sub}(\alpha) \cup \mathsf{sub}(\beta)\\
\mathsf{sub}(\langle\alpha\rangle) &:= \{\langle\alpha\rangle\} \cup \mathsf{sub}(\alpha)
\end{aligned}$$

B. MISSING PROOFS

Proof of Lemma 3.1. The fact that $\prec$ is a *quasi-order* (= reflexive and transitive) is immediate from its definition. To show that it is a *well* quasi-order, suppose we have an infinite sequence of configurations $\rho_1\rho_2\rho_3\cdots$. It is easy to see that it contains an infinite subsequence $\tau_1\tau_2\tau_3\cdots$ such that all its elements are of the form $\langle\alpha_0, (a_0, d), T\rangle$ with

- α_0 and a_0 fixed, and
- $h^T(d) = \mathcal{C}_0$ fixed,

where $h^T(d) = \{q \mid \langle q, d\rangle \in T\}$. This is because we can see each of these elements as a finite *coloring*, and apply the pigeonhole principle on the infinite set $\{\rho_i\}_i$.

Consider then the function $g^T : \wp(Q) \to \mathbb{N}$, such that $g^T(\mathcal{C}) = \#\{d \mid \mathcal{C} = h^T(d)\}$ (we can think of g^T as a tuple of $\mathbb{N}^{|\wp(Q)|}$). Assume the relation $\prec^\dagger$ defined as $T \prec^\dagger T'$ iff $g^T(\mathcal{C}) \leq g^{T'}(\mathcal{C})$ for all $\mathcal{C}$. By Dickson's Lemma $\prec^\dagger$ is a wqo, and then there are two $\tau_i = \langle\alpha_0, (s_0, d_i), T_i\rangle$, $\tau_j = \langle\alpha_0, (s_0, d_j), T_j\rangle$, $i < j$ such that $T_i \prec^\dagger T_j$. For each $\mathcal{C} \subset Q$, there exists an injective mapping $f_\mathcal{C} : \{d \mid h^{T_i}(d) = \mathcal{C}\} \to \{d \mid h^{T_j}(d) = \mathcal{C}\}$, as the latter set is bigger than the former by $\prec^\dagger$. We define the desired injection f as the (disjoint) union of all $f_\mathcal{C}$'s. In the case $h^T(d_i) = h^T(d_j) = \emptyset$, we also define $f(d_i) = d_j$. Hence, $\tau_i \prec \tau_j$. □

Proof of Proposition 3.4. The fact that $(A^*, \sqsubset)$ is a wqo is given by Higman's Lemma. The rdc property with

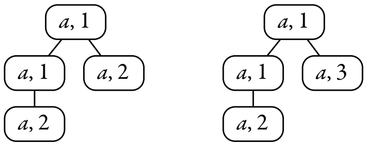


Figure 4: Two indistinguishable data trees for ATRA.

respect to $\rightarrow_2$ is straightforward. Let

$$\bar{u}'\,a'\,\bar{v}' \quad \sqsubset \quad \bar{u}\,a\,\bar{v} \quad \rightarrow_2 \quad \bar{u}\,\bar{b}\,\bar{v}$$

with $\bar{u}' \sqsubset \bar{u}$, $a' \leq a$, $\bar{v}' \sqsubset \bar{v}$, and $a \rightarrow_1 b_z$ for all $z \in [1..|\bar{b}|]$. As $a' \leq a$ and $\leq$ is rdc with $\rightarrow_1$, one possibility is that for each $a \rightarrow_1 b_z$ we can apply a $a' \rightarrow_1 b'_z$ with $b'_z \leq b_z$. In this case we obtain $\bar{u}'a\bar{v}' \rightarrow_2 \bar{u}'\bar{b}'\bar{v}' \sqsubset \bar{u}\bar{b}\bar{v}$. The only case left to analyze is when, for some $a \rightarrow_1 b_z$ the compatibility is reflexive, that is, $a' \leq b_z$. But then we take a reflexive compatibility as well, the reader can check that in this case $\bar{u}'a'\bar{v}' \sqsubset \bar{u}\bar{b}\bar{v}$.

Finally, in the case where a has no pre-image is only easier as we can take the exact same element. □

Proof of Lemma 4.2. Any $\overset{t}{\rightarrowtail}$ step shot by a $\rightarrow_\varepsilon$, $\rightarrow_\triangledown$, or $\rightarrow_\triangleright$ from a node configuration ρ leading to τ can be reproduced by $\underset{\cdot}{\overset{t}{\rightarrowtail}}$ from an isomorphic copy ρ' of ρ leading to an isomorphic copy of τ which is in the same relation to ρ' as τ is to ρ. This can be safely done without any collateral effects as the executions of two different node configurations $\langle p, \ldots \rangle$, $\langle p', \ldots \rangle$ have no interference one with another and are completely independent. The fact of whether one node configuration takes a data value that happens to be equal or not to a data value of another configuration no impact whatsoever in the execution. It suffices to examine the definition to see that $\underset{\cdot}{\overset{t}{\rightarrowtail}}$ depends on the transitions $\rightarrow_\triangledown$, $\rightarrow_\triangleright$ and $\rightarrow_\varepsilon$ of the *node* configurations. □

C. EXPRESSIVITY

Proof of Proposition 3.1, item (a). We show an example of the expressiveness that guess adds to ATRA. We force two incomparable nodes to have the same data value without any further data constraint. Note that this datum does not necessarily has to appear at some common ancestor of the nodes. Consider the ATRA(guess) defined over $\Sigma = \{a\}$ with

$$\begin{array}{lll} \delta(q_0) = \mathsf{guess}(q_1), & \delta(q_1) = \triangledown q_2, & \delta(q_2) = q_3 \wedge q_4, \\ \delta(q_3) = \triangledown q_5, & \delta(q_4) = \triangleright q_5, & \delta(q_5) = \mathsf{eq}. \end{array}$$

Consider the two data trees of Fig 4. It is easy to see that for any ATRA, either both are accepted, or both rejected. However, the ATRA(guess) we just built distinguishes them. □