An Automata-Theoretic Approach to Regular XPath*

Diego Calvanese[1], *Giuseppe De Giacomo*[2], *Maurizio Lenzerini*[2], and *Moshe Y. Vardi*[3]

[1] KRDB Research Centre, Free University of Bozen-Bolzano, Italy
calvanese@inf.unibz.it
[2] Dipartimento di Informatica e Sistemistica, SAPIENZA Università di Roma, Italy
degiacomo,lenzerini@dis.uniroma1.it
[3] Department of Computer Science
Rice University, P.O. Box 1892, Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu

In this paper we present Regular XPath (RXPath), which is a natural extension of XPath with regular expressions over paths that has the same computational properties as XPath: linear-time query evaluation and exponential-time reasoning. To establish these results, we devise a unifying automata-theoretic framework based on two-way weak alternating tree automata. Specifically, we consider automata that have infinite runs on finite trees. This enables us to leverage and simplify existing automata-theoretic machinery and develop algorithms both for query evaluation and for reasoning over queries. With respect to the latter problem, we consider RXPath as a constraint language, and study constraint satisfiability, and query satisfiability and containment under constraints in the setting of RXPath.

1 Introduction

XML[4] has become the standard language for semistructured data, and the last few years have witnessed a strong interest in reasoning about XML queries and integrity constraints. From a conceptual point of view, an XML document can be seen as a finite node-labeled tree, and several formalisms have been proposed as query languages over XML documents. A common feature of many of these language is the use of regular path expressions to navigate through XML documents, and *XPath* is a popular language for such navigation [9].

This paper introduces a powerful extension of *CoreXPath*, called *RXPath*, for expressing XML queries. Our language is inspired by the work carried out in the last few years on extensions of *CoreXPath* [29, 21]. In particular, we extend *CoreXPath* with nominals, as well as with regular path expressions over XML trees, expressed as two-way regular expressions over *XPath* axes. Our language is essentially Regular *XPath* [29], extended with nominals. A nominal is a formalism for denoting a single node in a document, similarly to XML global identifiers built through the construct `ID`. The power of our language in expressing paths is the one of Propositional Dynamic Logic (PDL) [15] extended with converse, nominals, and deterministic programs. This

* A preliminary version of this paper, dealing with *RXPath* satisfiability only, has been presented at the 2008 Workshop on Logic in Databases (LID 2008).

[4] http://www.w3.org/TR/REC-xml/

combination of path-forming constructs results in one of the most expressive languages ever considered for specifying structural queries over XML documents.

We describe in this paper a comprehensive automata-theoretic framework for evaluating and reasoning about *RXPath*. Our framework is based on *two-way weak alternating tree automata*, denoted 2WATA. The use of automata-theoretic techniques in the context of *XPath* is not new. For example, tree walking automata are used in [29] to characterize the expressive power of Regular *XPath*, while bottom-up tree automata are used there to derive an algorithm for testing containment of Regular *XPath* queries. In contrast, here we show that 2WATA provide a *unifying* formalism, as they enable us to both derive a linear-time algorithm for the evaluation of *RXPath* queries and an exponential-time algorithm for testing containment of *RXPath* queries.

Our automata-theoretic approach is based on techniques developed in the context of program logics [19, 30]. Here, however, we leverage the fact that we are dealing with finite trees, rather than the infinite trees used in the program-logics context. Indeed, the automata-theoretic techniques used in reasoning about infinite trees are notoriously difficult [25, 28] and have resisted efficient implementation. The restiction to finite trees here enables us to obtain much more feasible algorithmic approach. (A similar idea, focusing on query reasoning rather than query evaluation, has been pursued in [20, 16]. For the latter proposal, also an implementation is provided.) In particular, one can make use of symbolic techniques, at the base of modern model checking tools, for effectively querying and verifying XML documents. It is worth noting that while our automata run over finite trees they are allowed to have infinite runs. This separates 2WATA from the alternating tree automata used, e.g., in [11]. The key technical results here are that acceptance of trees by 2WATA can be decided in linear time, while nonemptiness of 2WATA can be decided in exponential time.

The first application of the automata-theoretic results is that *RXPath* queries can be evaluated in linear time; more precisely, the running time for query evaluation is a product of the size of the input tree and the size of the query. This extends the results in [17] of polynomial-time algorithms for the evaluation of *CoreXPath* queries. Note that [17] provide also results for full *XPath* that handles also data, hence goes beyond the core fragment considered here. Our result complements the one in [3], which considers an extension of *RXPath* with tests for equality between attributes at the end of two paths, and provides a query evaluation algorithm that is linear time in the size of the input tree but exponential in the size of the query.

The second application is to *RXPath* reasoning. There has been a lot of focus on testing containment of *XPath* queries, cf. [26]. Instead, we focus here on the more general problem of satisfying constraints over XML documents. Specifically, we consider here structural constraints [13]. Structural constraints are those imposing a certain form on the trees corresponding to the documents, with no explicit reference to values associated with nodes. Notable examples of formalisms allowing for expressing such constraints are DTDs (see footnote 4 and [6]), and XML Schema[5] [2]. We show how we can use *RXPath* to express such constraints. and then show that satisfiability of *RXPath* constraints can be checked in exponential time using our automata-theoretic results. The exponential decidability result for *RXPath* constraint satisfiability is not surpris-

[5] `http://www.w3.org/TR/xmlschema-0/` and `http://.../xmlschema-1/`

ing as this problem can be reduced to the satisfiability problem for *Repeat-Converse-Deterministic PDL (repeat-CDPDL)* a well-known variant of PDL, which can be solved using the automata-theoretic techniques of [30]. As noted above, however, those techniques are quite heavy and so far resisted implementation.

We also show that query satisfiability and query containment for *RXPath* can be reduced to checking satisfiability of *RXPath* constraints, thus enabling us to take advantage of the techniques developed for constraint satisfiability. Note that most previous results on this topic (see, e.g., [8]) refer to query languages that are either less expressive than *RXPath*, or are intended for semi-structured data modeled as graph-like structures, rather than XML trees.

2 Regular XPath

Following [21, 22], we formalize XML documents as finite sibling trees, which are tree like structures, whose nodes are linked to each other by two relations: the child relation, connecting each node with its children in the tree; and the immediate-right-sibling relation, connecting each node with its sibling immediately to the right in the tree, such a relation models the order between the children of the node in an XML documents. Each node of the sibling tree is labeled by (possibly many) elements of a set of atomic propositions Σ. We consider the set Σ to be partitioned into Σ_a and Σ_{id}. The set Σ_a is a set of atomic propositions that represent either XML tags or XML attribute-value pairs. On the other hand, Σ_{id} is a set of special propositions representing (node) *identifiers*, i.e., that are true in (i.e., that label) exactly a single node of the XML document. Such identifiers are essentially an abstraction of the XML identifiers built through the construct `ID` (see footnote 4), though a node can have multiple identifiers in our case. Observe that sibling trees are more general than XML documents since they would allow the same node to be labeled by several tags. It is easy to impose *RXPath* constraints (see later) that force propositions representing tags to be disjoint if needed.

A *sibling tree* is a pair $T_s = (\Delta^{T_s}, \cdot^{T_s})$, where Δ^{T_s} is a tree[6] and $\cdot^{T_s}$ is an interpretation function that assigns to each atomic symbol $A \in \Sigma_a$ a set A^{T_s} of nodes of Δ^{T_s}, to each identifier Id a singleton Id^{T_s} containing one node of Δ^{T_s}, and that interprets the axis relations in the obvious way, namely:

$$\begin{aligned}
\texttt{child}^{T_s} &= \{(z, z{\cdot}i) \mid z, z{\cdot}i \in \Delta^{T_s}\} \\
\texttt{right}^{T_s} &= \{(z{\cdot}i, z{\cdot}(i{+}1)) \mid z{\cdot}i, z{\cdot}(i{+}1) \in \Delta^{T_s}\}
\end{aligned}$$

As in [21, 22], we focus on a variant of *XPath* that allows for full regular expressions over the *XPath* axes. In fact, we make it explicit that such a variant of *XPath* is tightly related to Propositional Dynamic Logic (PDL) [1, 15], and adopt the PDL syntax to express node and path expressions.

[6] A (finite) tree is a non-empty (finite) set Δ of words over $\mathbb{N}$, such that if $x{\cdot}i \in \Delta$, where $x \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then also $x \in \Delta$, and if $i > 1$, then also $x{\cdot}(i{-}1) \in \Delta$. By convention we take $x{\cdot}0 = x$, and $x{\cdot}i{\cdot}{-}1 = x$. A *(finite) labeled tree* over an alphabet $\mathcal{L}$ of labels is a pair $T = (\Delta^T, \ell^T)$, where Δ^T is a (finite) tree and the labeling $\ell^T : \Delta^T \to \mathcal{L}$ is a mapping assigning to each node $x \in \Delta^T$ a label $\ell^T(x)$ in $\mathcal{L}$.

$$
\begin{array}{lll}
(\langle P\rangle\varphi)^{T_s} & = \{z \mid \exists z'.(z,z') \in P^{T_s} \wedge z' \in \varphi^{T_s}\} & (\varphi?)^{T_s} = \{(z,z) \mid z \in \varphi^{T_s}\} \\
([P]\varphi)^{T_s} & = \{z \mid \forall z'.(z,z') \in P^{T_s} \rightarrow z' \in \varphi^{T_s}\} & (P_1;P_2)^{T_s} = P_1^{T_s} \circ P_2^{T_s} \\
(\neg\varphi)^{T_s} & = \Delta^T \setminus \varphi^{T_s} & (P_1 \cup P_2)^{T_s} = P_1^{T_s} \cup P_2^{T_s} \\
(\varphi_1 \wedge \varphi_2)^{T_s} & = \varphi_1^{T_s} \cap \varphi_2^{T_s} & (P^*)^{T_s} = (P^{T_s})^* \\
(\varphi_1 \vee \varphi_2)^{T_s} & = \varphi_1^{T_s} \cup \varphi_2^{T_s} & (P^-)^{T_s} = \{(z',z) \mid (z,z') \in P^{T_s}\}
\end{array}
$$

Fig. 1. Semantics of node and path expressions

RXPath expressions are of two sorts: *node expressions*, denoted by φ, and *path expressions*, denoted by P, defined by the following syntax (we omit parentheses):

$$
\begin{array}{lcl}
\varphi & \longrightarrow & A \mid Id \mid \langle P\rangle\varphi \mid [P]\varphi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \\
P & \longrightarrow & \texttt{child} \mid \texttt{right} \mid \varphi? \mid P_1;P_2 \mid P_1 \cup P_2 \mid P^* \mid P^-
\end{array}
$$

where $A \in \Sigma_a$, $Id \in \Sigma_{id}$, and `child` and `right` denote the two main *XPath* axis relations. We consider the other XPath axis relations `parent` and `left` as abbreviations for $\texttt{child}^-$ and $\texttt{right}^-$, respectively. Also, we use the usual abbreviations, including `true`, `false`, and $\varphi_1 \rightarrow \varphi_2$.

Given a sibling tree $T_s = (\Delta^{T_s}, \cdot^{T_s})$, we extend the interpretation function $\cdot^{T_s}$ to arbitrary node and path expressions as shown in Figure 1, where we have used the standard notions of chaining ($\cdot \circ \cdot$) and reflexive-transitive closure ($\cdot^*$) over binary relations. Note that, $[P]\varphi$ is equivalent to $\neg\langle P\rangle\neg\varphi$.

To develop our techniques for inference on *RXPath*, it is convenient to consider an additional axis `fchild`, connecting each node to its first child only, interpreted as

$$\texttt{fchild}^{T_s} = \{(z, z{\cdot}1) \mid z, z{\cdot}1 \in \Delta^{T_s}\}$$

Using `fchild`, we can thus re-express the `child` axis as $\texttt{fchild};\texttt{right}^*$. In this way, we can view sibling trees, which are unranked, as binary trees (see Section 4).

We say that a path expression is *normalized* if it is expressed by making use of `fchild` and `right` only, if $\cdot^-$ is pushed inside as much as possible, in such a way that it appears only in front of `fchild` and `right` only, and if all node expressions occurring in it are normalized. A node expression is normalized if all path expressions occurring in it are normalized, and if it is in *negation normal form*, i.e., negation is pushed inside as much as possible, in such a way that it appears only in front of atomic symbols.

RXPath expressions can be used to express queries on XML documents. An *RXPath (unary) query* is an *RXPath* node expression φ that, when evaluated over a sibling tree T_s, returns the set of nodes φ^{T_s}. We also consider *RXPath binary queries*, where such a query is an *RXPath* path expression P that, when evaluated over a sibling tree T_s, returns the set of pairs of nodes P^{T_s}. We will address the problems of query evaluation and of query satisfiability by means of automata-theoretic techniques, presented in the next section. A further use of *RXPath* expressions is to specify constraints, which will be dealt with in Section 5, resorting again to automata.

3 Two-way Weak Alternating Tree Automata

We consider a variant of two-way alternating automata [27] that run, possibly infinitely, on finite labeled trees Specifically, alternating tree automata generalize nondetermin-

istic tree automata, while two-way tree automata generalize ordinary tree automata by being allowed to traverse the tree both upwards and downwards. Formally, let $\mathcal{B}^+(I)$ be the set of positive Boolean formulae over a set I, built inductively by applying $\wedge$ and $\vee$ starting from **true**, **false**, and elements of I. For a set $J \subseteq I$ and a formula $\varphi \in \mathcal{B}^+(I)$, we say that J *satisfies* φ if assigning **true** to the elements in J and **false** to those in $I \setminus J$, makes φ true. For a positive integer k, let $[1..k] = \{1, \ldots, k\}$ and $[-1..k] = \{-1, 0, 1, \ldots, k\}$. For integers i, j, with $i \leq j$, let $[i..j] = \{i, \ldots, j\}$. A *two-way weak alternating tree automaton* (2WATA) running over labeled trees all of whose nodes have at most k successors, is a tuple $\mathbf{A} = (\mathcal{L}, S, s_0, \delta, \alpha)$, where $\mathcal{L}$ is the alphabet of tree labels, S is a finite set of states, $s_0 \in S$ is the initial state, $\delta : S \times \mathcal{L} \to \mathcal{B}^+([-1..k] \times S)$ is the transition function, and α is the accepting condition discussed below.

The transition function maps a state $s \in S$ and an input label $a \in \mathcal{L}$ to a positive Boolean formula over $[-1..k] \times S$. Intuitively, if $\delta(s, a) = \varphi$, then each pair (c', s') appearing in φ corresponds to a new copy of the automaton going to the direction suggested by c' and starting in state s'. For example, if $k = 2$ and $\delta(s_1, a) = ((1, s_2) \wedge (1, s_3)) \vee ((-1, s_1) \wedge (0, s_3))$, when the automaton is in the state s_1 and reads the node x labeled by a, it proceeds either by sending off two copies, in the states s_2 and s_3 respectively, to the first successor of x (i.e., $x{\cdot}1$), or by sending off one copy in the state s_1 to the predecessor of x (i.e., $x{\cdot}{-1}$) and one copy in the state s_3 to x itself (i.e., $x{\cdot}0$).

A run of a 2WATA is obtained by resolving all existential choices. The universal choices are left, which gives us a tree. Because we are considering two-way automata, runs can start at arbitrary tree nodes, and need not start at the root. Formally, a run of a 2WATA $\mathbf{A}$ over a labeled tree $T = (\Delta^T, \ell^T)$ from a node $x_0 \in \Delta^T$ is a finite $\Delta^T \times S$-labeled tree $R = (\Delta^R, \ell^R)$ satisfying:

1. $\varepsilon \in \Delta^R$ and $\ell^R(\varepsilon) = (x_0, s_0)$.
2. Let $\ell^R(r) = (x, s)$ and $\delta(s, \ell^T(x)) = \varphi$. Then there is a (possibly empty) set $S = \{(c_1, s_1), \ldots, (c_n, s_n)\} \subseteq [-1..k] \times S$ such that S satisfies φ, and for each $i \in [1..n]$, we have that $r{\cdot}i \in \Delta^R$, $x{\cdot}c_i \in \Delta^T$, and $\ell^R(r{\cdot}i) = (x{\cdot}c_i, s_i)$.

Intuitively, a run R keeps track of all transitions that the 2WATA $\mathbf{A}$ performs on a labeled input tree T: a node r of R labeled by (x, s) describes a copy of $\mathbf{A}$ that is in the state s and is reading the node x of T. The successors of r in the run represent the transitions made by the multiple copies of $\mathbf{A}$ that are being sent off either upwards to the predecessor of x, downwards to one of the successors of x, or to x itself.

A 2WATA is called "weak" due to the specific form of the acceptance condition α. Specifically, $\alpha \subseteq S$, and there exists a partition of S into disjoint sets, S_i, such that for each set S_i, either $S_i \subseteq \alpha$, in which case S_i is an *accepting set*, or $S_i \cap \alpha = \emptyset$, in which case S_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the S_i's such that, for each $s \in S_i$ and $s' \in S_j$ for which s' occurs in $\delta(s, a)$, for some $a \in \mathcal{L}$, we have $S_j \leq S_i$. Thus, transitions from a state in S_i lead to states in either the same S_i or a lower one. It follows that every infinite path of a run of a 2WATA ultimately gets "trapped" within some S_i. The path is *accepting* if and only if S_i is an accepting set. A run (T_r, r) is *accepting* if all its infinite paths are accepting. A 2WATA $\mathbf{A}$ *accepts* a labeled tree T from a node $x_0 \in \Delta^T$ if there exists an accepting run of $\mathbf{A}$

over T from x_0. The *language* $\mathscr{L}(\mathbf{A})$ accepted by $\mathbf{A}$ is the set of trees that $\mathbf{A}$ accepts from the root ε.

3.1 The Acceptance Problem

Given a 2WATA $\mathbf{A} = (\mathcal{L}, S, s_0, \delta, \alpha)$, a labeled tree $T = (\Delta^T, \ell^T)$, and a node $x_0 \in \Delta^T$, we'd like to know whether $\mathbf{A}$ accepts T from x_0. This is called the *acceptance problem*. We follow here the approach of [19], and solve the acceptance problem by first taking a product $\mathbf{A} \times T_{x_0}$ of $\mathbf{A}$ and T from x_0. This product is an alternating automaton over a one letter alphabet $\mathcal{L}_0$, consisting of a single letter, say a. This product automaton simulates a run of $\mathbf{A}$ on T from x_0. The product automaton is $\mathbf{A} \times T_{x_0} = (\mathcal{L}_0, S \times \Delta^T, (s_0, x_0), \delta', \alpha \times \Delta^T)$, where δ' is defined as follows:

- $\delta'((s,x),a) = \Theta_x(\delta(s, \ell^T(x)))$, where Θ_x is the substitution that replaces a pair (c,t), by the pair $(t, x{\cdot}c)$ if $x{\cdot}c \in \Delta^T$, and by **false** otherwise.

Note that the size of $\mathbf{A} \times T_{x_0}$ is simply the product of the size of $\mathbf{A}$ and the size of T. Note also that $\mathbf{A} \times T_{x_0}$ can be viewed an a weak alternating word automaton running over the infinite word a^ω, as by taking the product with T we have eliminated all directions.

We can now state the relationship between $\mathbf{A} \times T_{x_0}$ and $\mathbf{A}$, which is essentially a restatement of Proposition 3.2 in [19].

Proposition 1. $\mathbf{A}$ *accepts* T *from* x_0 *iff* $\mathbf{A} \times T_{x_0}$ *accepts* a^ω.

The advantage of Proposition 1 is that it reduces the acceptance problem to the question of whether $\mathbf{A} \times T$ accepts a^ω. This problem is referred to in [19] as the "one-letter nonemptiness problem". It is shown there that this problem can be solved in time that is linear in the size of $\mathbf{A} \times T_{x_0}$ by an algorithm that imposes an evaluation of and-or trees over a decomposition of the automaton state space into maximal strongly connected components. The result in [19] is actually stronger; the algorithm there computes in linear time the set of initial states from which the automaton accepts a^ω. We therefore obtain the following result about the acceptance problem.

Proposition 2. *Given a 2WATA* $\mathbf{A}$ *and a labeled tree* T*, we can compute in time that is linear in the product of the sizes of* $\mathbf{A}$ *and* T *the set of nodes* x_0 *such that* $\mathbf{A}$ *accepts* T *from* x_0*.*

3.2 The Nonemptiness Problem

The *nonemptiness problem* for 2WATAs consists in determining, for a given 2WATA $\mathbf{A}$ whether it accepts some tree T from ε. This problem is solved in [30] for 2WATAs (actually, for a more powerful automata model) over infinite trees, using rather sophisticated automata-theoretic techniques. Here we solve this problem over finite trees, which requires less sophisticated techniques, which are much easier to implement.

In order to decide non-emptiness of 2WATAs, we resort to a conversion to standard one-way nondeterministic tree automata [10]. A one-way nondeterministic tree automaton (NTA) is a tuple $\mathbf{A} = (\mathcal{L}, S, s_0, \delta)$, analogous to a 2WATA, except that (*i*) the acceptance condition α is empty and has been dropped from the tuple, (*ii*) the directions

-1 and 0 are not used in δ and, (*iii*) for each state $s \in S$ and letter $a \in \mathcal{L}$, the positive Boolean formula $\delta(s, a)$, when written in DNF, does not contain a disjunct with two distinct atoms (c, s_1) and (c, s_2) with the same direction c. In other words, each disjunct corresponds to sending at most one "subprocess" in each direction. While for 2WATAs we have separate input tree and run tree, for NTAs we can assume that the run of the automaton over an input tree $T = (\Delta^T, \ell^T)$ is an S-labeled tree $R = (\Delta^T, \ell^R)$, which has the same underlying tree as T, and thus is finite, but is labeled by states in S. Nonemptiness of NTAs is known to be decidable in linear time [12].

It remains to describe the translation of 2WATAs to NTAs. Given a 2WATA $\mathbf{A}$ and an input tree T as above, a *strategy for* $\mathbf{A}$ *on* T is a mapping $\tau : \Delta^T \to 2^{S \times [-1..k] \times S}$. Thus, each label in a strategy is an edge-$[-1..k]$-labeled directed graph on S. Intuitively, each label is a set of transitions. For each label $\zeta \subseteq S \times [-1..k] \times S$, we define $state(\zeta) = \{u : (u, i, v) \in \zeta\}$, i.e., $state(\zeta)$ is the set of sources in the graph ζ. In addition, we require the following: (1) $s_0 \in state(\tau(\varepsilon))$, (2) for each node $x \in \Delta^T$ and each state $s \in state(\tau(x))$, the set $\{(c, s') : (s, c, s') \in \tau(x)\}$ satisfies $\delta(s, \ell^T(x))$ (thus, each label can be viewed as a strategy of satisfying the transition function), and (3) for each node $x \in \Delta^T$, and each edge $(s, i, s') \in \tau(x)$, we have that $s' \in state(\tau(x{\cdot}i))$.

A *path* β in the strategy τ is a maximal sequence $(u_0, s_0), (u_1, s_1), \ldots$ of pairs from $\Delta^T \times S$ such that $u_0 = \varepsilon$ and, for all $i \geq 0$, there is some $c_i \in [-1..k]$ such that $(s_i, c_i, s_{i+1}) \in \tau(u_i)$ and $u_{i+1} = u_i{\cdot}c_i$. Thus, β is obtained by following transitions in the strategy. The path β is accepting if the path $s_0, s_1, \ldots$ is accepting. The strategy τ is *accepting* if all its paths are accepting.

Proposition 3 ([30]). *A 2WATA* $\mathbf{A}$ *accepts an input tree* T *from* ε *iff* $\mathbf{A}$ *has an accepting strategy tree for* T.

We have thus succeeded in defining a notion of run for alternating automata that will have the same tree structure as the input tree. We are still facing the problem that paths in a strategy tree can go both up and down. We need to find a way to restrict attention to uni-directional paths. For this we need an additional concept.

An *annotation* for $\mathbf{A}$ on T with respect to a strategy τ is a mapping $\eta : \Delta^T \to 2^{S \times \{0,1\} \times S}$. Thus, each label in an annotation is an edge-$\{0, 1\}$-labeled directed graph on S. We assume that edge labels are unique; that is, a graph cannot contain both triples $(s, 0, s')$ and $(s, 1, s')$. We require η to satisfy some closure conditions for each node $x \in \Delta^T$. Intuitively, these conditions say that η contains all relevant information about finite paths in τ. Thus, an edge (s, c, s') describes a path from s to s', where $c = 1$ if this path goes through α. The conditions are: (a) if $(s, c, s') \in \eta(x)$ and $(s', c', s'') \in \eta(x)$, then $(s, c'', s'') \in \eta(x)$ where $c'' = \max\{c, c'\}$, (b) if $(s, 0, s') \in \tau(x)$ then $(s, c, s') \in \eta(x)$, where $c = 1$ if $s' \in \alpha$ and $c = 0$ otherwise, (c) if $x = y{\cdot}i$, $(s, -1, s') \in \tau(x)$, $(s', c, s'') \in \eta(y)$, and $(s'', i, s''') \in \tau(x)$, then $(s, c', s''') \in \eta(x)$, where $c' = 1$ if either $s' \in \alpha$, $c = 1$, or $s''' \in \alpha$, and $c' = 0$ otherwise, and (d) if $y = x{\cdot}i$, $(s, i, s') \in \tau(x)$, $(s', c, s'') \in \eta(y)$, and $(s'', -1, s''') \in \tau(y)$, then $(s, c', s''') \in \eta(x)$, where $c' = 1$ if $s \in \alpha$, $c = 1$, or $s''' \in \alpha$, and $c' = 0$ otherwise. The annotation η is *accepting* if for every node $x \in \Delta^T$ and state $s \in S$, if $(s, c, s) \in \eta(x)$, then $c = 1$. In other words, η is accepting if all cycles visit accepting states.

Proposition 4 ([30]). *A 2WATA* $\mathbf{A}$ *accepts an input tree* T *from* ε *iff* $\mathbf{A}$ *has a strategy tree* τ *on* T *and an accepting annotation* η *of* τ.

Consider now *annotated trees* $(\Delta^T, \ell^T, \tau, \eta)$, where τ is a strategy tree for $\mathbf{A}$ on (Δ^T, ℓ^T) and η is an annotation of τ. We say that $(\Delta^T, \ell^T, \tau, \eta)$ is *accepting* if η is accepting.

Theorem 5. *Let* $\mathbf{A}$ *be a 2WATA. Then there is an NTA* $\mathbf{A}_n$ *such that* $\mathscr{L}(\mathbf{A}) = \mathscr{L}(\mathbf{A}_n)$. *The number of states of* $\mathbf{A}_n$ *is exponential in the number of states of* $\mathbf{A}$.

The key feature of the state space of $\mathbf{A}_n$ is the fact that states are pairs consisting of subsets of S and $S \times \{0,1\} \times S$. Thus, a set of states of $\mathbf{A}_n$ can be described by a Boolean function on the domain S^3. Similarly, the transition function of $\mathbf{A}_n$ can also be described as a Boolean function. Such functions can be represented by BDDs [4], enabling a symbolic approach to nonemptiness testing of 2WATAs, as shown below. We note that the framework of [30] also converts a two-way alternating tree automaton (on infinite trees) to a nondeterministic tree automaton (on infinite trees). The state space of the latter, however, is considerably more complex than the one obtained here due to Safra's determinization construction. This makes it practically infeasible to apply the symbolic approach in the infinite-tree setting.

Theorem 6. *Given a 2WATA* $\mathbf{A}$ *with* n *states and an input alphabet with* m *elements, deciding nonemptiness of* $\mathbf{A}$ *can be done in time exponential in* n *and linear in* m.

As shown in [21] (see also Section 5 for dealing with identifiers), reasoning over *RXPath* formulas can be reduced to checking satisfiability in *Propositional Dynamic Logics (PDLs)*. Specifically, one can resort to *Repeat-Converse-Deterministic PDL (repeat-CDPDL)*, a variant of PDL that allows for expressing the finiteness of trees and for which satisfiability is EXPTIME-complete [30]. This upper bound, however, is established using sophisticated infinite-tree automata-theoretic techniques (cf., e.g., [24]), which so far have resisted attempts at practically efficient implementation [25, 28], due to the use of Safra's determinization construction [23] and parity games [18]. The main advantage of our approach here is that we use only automata on finite trees, which require a much "lighter" automata-theoretic machinery. As noted in Theorem 6, nonemptiness for 2WATA can be tested in time that is exponential in the number of states and linear in the size of the alphabet. One can show that our 2WATA-based decision procedure can be implemented using a *symbolic* approach, which has the potential to be capable of handling automata with large states spaces [5].

4 2WATAs for *RXPath* Query Evaluation

We address now the problem of evaluating *RXPath* queries by means of 2WATAs. To do so, we first represent sibling trees as binary trees, and then encode the problem of evaluating an *RXPath* query φ into the acceptance problem for a 2WATA $\mathbf{A}_\varphi^{wf}$ whose number of states is linear in φ. This allows us to establish a tight complexity bound for *RXPath* query evaluation.

We work on binary trees. In order for such trees to represent sibling trees, we make use of special labels ifc, irs, hfc, hrs, where ifc (resp., irs) are used to keep track of whether a node *is the first child* (resp., *is the right sibling*) of its predecessor, and hfc (resp., hrs) are used to keep track of whether a node *has a first child* (resp., *has a right sibling*). Formally, we consider binary trees whose nodes are labeled with subsets of $\Sigma \cup \{ifc, irs, hfc, hrs\}$. We call such a tree $T = (\Delta^T, \ell^T)$ *well-formed* if it satisfies the following conditions:

- For each node x of T, if $\ell^T(x)$ contains hfc, then $x{\cdot}1$ is meant to represent the `fchild` successor of x and hence $\ell^T(x{\cdot}1)$ contains ifc but not irs. Similarly, if $\ell^T(x)$ contains hrs, then $x{\cdot}2$ is meant to represent the `right` successor of x and hence $\ell^T(x{\cdot}2)$ contains irs but not ifc.
- The label $\ell^T(\varepsilon)$ of the root of T contains neither ifc, nor irs, nor hrs. In this way, we restrict the root of T so as to represent the root of a sibling tree.
- For each $Id \in \Sigma_{id}$, there is at most one node x of T with $Id \in \ell^T(x)$.

A sibling tree $T_s = (\Delta^{T_s}, \cdot^{T_s})$, induces a well-formed binary tree $\pi_b(T_s)$. To define $\pi_b(T_s) = (\Delta^T, \ell^T)$, we define, by induction on Δ^{T_s}, both a mapping π_b from Δ^{T_s} to nodes of a binary tree, and the labeling of such nodes with ifc, irs, hfc, and hrs as follows:

- $\pi_b(\varepsilon) = \varepsilon$;
- $\pi_b(x{\cdot}1) = \pi_b(x){\cdot}1$, for each node $x{\cdot}1 \in \Delta^{T_s}$; moreover, $hfc \in \ell^T(\pi_b(x))$ and $ifc \in \ell^T(\pi_b(x){\cdot}1)$;
- $\pi_b(x{\cdot}(n{+}1)) = \pi_b(x{\cdot}n){\cdot}2$, for each node $x{\cdot}(n{+}1) \in \Delta^{T_s}$, with $n \geq 1$; moreover, $hrs \in \ell^T(\pi_b(x{\cdot}n))$ and $irs \in \ell^T(\pi_b(x{\cdot}n){\cdot}2)$.

Then, we take Δ^T to be the range of π_b, and we complete the definition of the labeling $\ell^T(\pi_b(x))$, for each node $x \in \Delta^{T_s}$, as follows: $A \in \ell^T(\pi_b(x))$ iff $x \in A^{T_s}$, for each $A \in \Sigma_a$, and $Id \in \ell^T(\pi_b(x))$ iff $x \in Id^{T_s}$, for each $Id \in \Sigma_{id}$. Notice that, since in T_s Id is interpreted as a singleton, T is well-formed.

To simplify the use of automata-theoretic techniques, we assume in the following that (normalized) path expressions are represented by means of finite automata rather than regular expressions. More precisely, a normalized path expression is represented as a finite automaton on finite words (NFA) $P = (\Theta, Q, q_0, \varrho, F)$, in which the alphabet Θ is constituted by `fchild`, `fchild`$^-$, `right`, `right`$^-$ and by node expressions followed by ?. The semantics of a path expression represented in such a way is the adaptation of the semantics for path expressions as given in Section 2, when we view them as regular expressions, with ;, $\cup$, and * representing respectively the concatenation, union, and Kleene star operators: a path expression P_N represented as an NFA denotes the same set of pairs of nodes as a path expression P_R represented as a regular expression and defining the same language as P_N, where the correspondence is applied inductively to the path expressions appearing in the node expressions of P_N (respectively P_R).

We need to make use of a notion of syntactic closure, similar to that of Fisher-Ladner closure of a formula of PDL [15]. We need first to define the closure of path expressions: given a path expression $P = (\Theta, Q, q_0, \varrho, F)$, we denote with P_q the path expression $P_q = (\Theta, Q, q, \varrho, F)$ obtained from P by making $q \in Q$ the initial state. The *closure*

if $\psi \in CL(\varphi)$ then $nnf(\neg\psi) \in CL(\varphi)$ (if ψ is not of the form $\neg\psi'$)
if $\neg\psi \in CL(\varphi)$ then $\psi \in CL(\varphi)$
if $\psi_1 \wedge \psi_2 \in CL(\varphi)$ then $\psi_1,\ \psi_2 \in CL(\varphi)$
if $\psi_1 \vee \psi_2 \in CL(\varphi)$ then $\psi_1,\ \psi_2 \in CL(\varphi)$
if $\langle P\rangle\psi \in CL(\varphi)$ then $\psi \in CL(\varphi)$, and $\langle P_q\rangle\psi \in CL(\varphi)$ for each $P_q \in CL(P)$
if $\langle P\rangle\psi \in CL(\varphi)$, where $P = (\Theta, Q, q_0, \varrho, F)$, then $\psi' \in CL(\varphi)$, for each $\psi'? \in \Theta$
if $[P]\psi \in CL(\varphi)$ then $\psi \in CL(\varphi)$, and $[P_q]\psi \in CL(\varphi)$ for each $P_q \in CL(P)$
if $[P]\psi \in CL(\varphi)$, where $P = (\Theta, Q, q_0, \varrho, F)$, then $\psi' \in CL(\varphi)$, for each $\psi'? \in \Theta$

Fig. 2. Closure of *RXPath* expressions

$CL(P)$ of P is the set $CL(P) = \{P_q \mid q \in Q\}$. The *syntactic closure* $CL(\varphi)$ of a node expression φ is defined inductively by asserting that $\{\varphi, \mathit{ifc}, \mathit{irs}, \mathit{hfc}, \mathit{hrs}\} \subseteq CL(\varphi)$, and by the rules in Figure 2, where $nnf(\neg\psi)$ denotes the negation normal form of $\neg\psi$.

Proposition 7. *Given a node expression φ, the cardinality of $CL(\varphi)$ is linear in the length of φ.*

Let φ be a normalized node expression. We first show how to construct a 2WATA $\mathbf{A}_\varphi$ that, when run over the binary tree corresponding to a sibling tree T_s, accepts exactly from the nodes corresponding to those in φ^{T_s}. The 2WATA $\mathbf{A}_\varphi = (\mathcal{L}, S_\varphi, s_\varphi, \delta_\varphi, \alpha_\varphi)$ is defined as follows.

- The alphabet is $\mathcal{L} = 2^{\Sigma'}$, i.e., all sets consisting of atomic symbols and the special symbols ifc, irs, hfc, hrs. This corresponds to labeling each node of the tree with a truth assignment to the atomic symbols, with information about the predecessor node, and with information about whether the children are significant.
- The set of states is $S_\varphi = CL(\varphi)$. Intuitively, when the automaton is in a state $\psi \in CL(\varphi)$ and visits a node x of the tree, this means that the automaton has to check that node expression ψ holds in x. When ψ is an atomic symbol α, i.e., an atomic proposition, an identifier, or one of the special symbols ifc, irs, hfc, hrs, this amounts to checking that the node label contains α.
- The initial state is $s_\varphi = \varphi$.
- The transition function δ_φ is defined as follows:
  1. For each $\lambda \in \mathcal{L}$, and each symbol $\alpha \in \Sigma \cup \{\mathit{ifc}, \mathit{irs}, \mathit{hfc}, \mathit{hrs}\}$, there are transitions
  $$\delta_\varphi(\alpha, \lambda) = \begin{cases} \textbf{true}, & \text{if } \alpha \in \lambda \\ \textbf{false}, & \text{if } \alpha \notin \lambda \end{cases} \qquad \delta_\varphi(\neg\alpha, \lambda) = \begin{cases} \textbf{true}, & \text{if } \alpha \notin \lambda \\ \textbf{false}, & \text{if } \alpha \in \lambda \end{cases}$$
  Such transitions check the truth value of atomic and special symbols and their negations in the current node of the tree.
  2. For each $\lambda \in \mathcal{L}$ and each $\psi_1, \psi_2 \in CL(\varphi)$, there are transitions
  $$\delta_\varphi(\psi_1 \wedge \psi_2, \lambda) = (0, \psi_1) \ \wedge \ (0, \psi_2)$$
  $$\delta_\varphi(\psi_1 \vee \psi_2, \lambda) = (0, \psi_1) \ \vee \ (0, \psi_2)$$
  Such transitions inductively decompose node expressions and move to appropriate states of the automaton to check the subexpressions.

3. For each $\lambda \in \mathcal{L}$ and each $\langle P \rangle \psi \in CL(\varphi)$, where $P = (\Theta, Q, q_0, \varrho, F)$, there is a transition $\delta_\varphi(\langle P \rangle \psi, \lambda)$ constituted by the disjunction of the following parts:

if	$q_0 \in F$	then	$(0, \psi)$
if	$q \in \varrho(q_0, \texttt{fchild})$	then	$(0, hfc) \wedge (1, \langle P_q \rangle \psi)$
if	$q \in \varrho(q_0, \texttt{right})$	then	$(0, hrs) \wedge (2, \langle P_q \rangle \psi)$
if	$q \in \varrho(q_0, \texttt{fchild}^-)$	then	$(0, ifc) \wedge (-1, \langle P_q \rangle \psi)$
if	$q \in \varrho(q_0, \texttt{right}^-)$	then	$(0, irs) \wedge (-1, \langle P_q \rangle \psi)$
if	$q \in \varrho(q_0, \psi'?)$	then	$(0, \psi') \wedge (0, \langle P_q \rangle \psi)$

Such transitions check step-by-step the existence of a path on the tree that conforms to the path expressions P and such that ψ holds at the ending node.

4. For each $\lambda \in \mathcal{L}$ and each $[P]\psi \in CL(\varphi)$, where $P = (\Theta, Q, q_0, \varrho, F)$, there is a transition $\delta_\varphi([P]\psi, \lambda)$ constituted by the conjunction of the following parts:

if	$q_0 \in F$	then	$(0, \psi)$
if	$q \in \varrho(q_0, \texttt{fchild})$	then	$(0, \neg hfc) \vee (1, [P_q]\psi)$
if	$q \in \varrho(q_0, \texttt{right})$	then	$(0, \neg hrs) \vee (2, [P_q]\psi)$
if	$q \in \varrho(q_0, \texttt{fchild}^-)$	then	$(0, \neg ifc) \vee (-1, [P_q]\psi)$
if	$q \in \varrho(q_0, \texttt{right}^-)$	then	$(0, \neg irs) \vee (-1, [P_q]\psi)$
if	$q \in \varrho(q_0, \psi'?)$	then	$(0, nnf(\neg\psi')) \vee (0, [P_q]\psi)$

Such transitions check step-by-step that for all paths on the tree that conform to the path expressions P we get that ψ holds at the ending node.

– The acceptance conditions α_φ is the set of all node expressions $[P]\psi \in CL(\varphi)$. Observe that a simple partition $S_\varphi = \cup_i S_i$ of the set of states resulting from the above transition function is the one that reflects the syntactic structure of φ, and that puts all literals, including the ones corresponding to the special labels, in a single element of the partition ordered below all other elements. Specifically, for each pair P, ψ for which $[P]\psi$ or $\langle P \rangle \psi$ appears explicitly in $nnf(\varphi)$, the set of node expressions $\{[P_q]\psi \mid P_q \in CL(P)\}$ form an element S_i of the partition; similarly, the set of node expressions $\{\langle P_q \rangle \psi \mid P_q \in CL(P)\}$ form another element S_j of the partition. All other states in S_φ form a singleton element of the partition, and sub-expressions are ordered below their containing expression. Note that all node expressions $\langle P \rangle \psi$ are in a rejecting set, which ensures that their satisfaction cannot be postponed indefinitely in an accepting run.

As for the size of $\mathbf{A}_\varphi$, by Proposition 7, from the above construction we get:

Proposition 8. *The number of states of $\mathbf{A}_\varphi$ is linear in the size of φ.*

Theorem 9. *Let φ be a normalized node expression, $\mathbf{A}_\varphi$ the 2WATA constructed above, T_s a sibling tree, and $\pi_b(T_s)$ the corresponding (well-formed) binary tree. Then a node x of T_s is in φ^{T_s} iff $\mathbf{A}_\varphi$ accepts $\pi_b(T_s)$ from $\pi_b(x)$.*

From Proposition 2 and Theorem 9, we immediately get our first main result.

Theorem 10. *Given a sibling tree T_s and an RXPath query φ, we can compute φ^{T_s} in time that is linear in the number of nodes of T_s (data complexity) and in the size of φ (query complexity).*

This technique can be used also to evaluate binary queries. Indeed, we can adorn (in linear time in the size of T_s) each node y of T_s with a unique identifier Id_y, obtaining a tree T'_s. Then, to evaluate the *RXPath* binary query P, we consider the unary queries $\varphi_{P,y} = \langle P \rangle Id_y$. The answer to P over T_s is simply $P^{T_s} = \bigcup_{y \in T_s} \{(x, y) \mid x \in \varphi_{P,y}^{T'_s}\}$, which, by Theorem 10, can be computed in quadratic time in the number of nodes of T_s and in linear time in the size of P.

5 Reasoning on *RXPath*

We start by introducing *RXPath root constraints*, which are node expressions intended to be true on the root of the document, and study the problem of satisfiability and implication of such constraints. Formally, the root constraint φ is *satisfied* in a sibling tree T_s if $\varepsilon \in \varphi^{T_s}$. A (finite) set Γ of *RXPath* root constraints is *satisfiable* if there exists a sibling tree T_s that satisfies all constraints in Γ. A set Γ of *RXPath* root constraints implies an *RXPath* root constraint φ, written $\Gamma \models \varphi$, if φ is satisfied in every sibling tree that satisfies all constraints in Γ. Note that unsatisfiability and implication of *RXPath* root constraints are mutually reducible to each other. Indeed Γ is unsatisfiable if and only if $\Gamma \models \mathtt{false}$. Also, $\Gamma \models \varphi$ if and only if $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable. Hence, in the following, we deal with satisfiability only.

In [21] it was shown that for *RXPath* root constraints without identifiers satisfiability is ExpTime-complete[7]. Here, as mentioned, we include special propositions Σ_{id} representing identifiers. However, the condition that a proposition is an identifier, i.e., denotes a singleton, can be expressed in *RXPath*. Indeed we can force a proposition A to be a singleton by using the root constraint N_A defined as follows, where the abbreviation $\mathbf{u}$ denotes the path expression $(\mathtt{fchild} \cup \mathtt{right})^*$:

$$\begin{aligned} N_A \;=\; & \langle \mathbf{u} \rangle A \wedge & (1)\\ & [\mathbf{u}]((\langle \mathtt{fchild}; \mathbf{u} \rangle A \rightarrow [\mathtt{right}; \mathbf{u}]\neg A) \wedge & (2)\\ & \quad (\langle \mathtt{right}; \mathbf{u} \rangle A \rightarrow [\mathtt{fchild}; \mathbf{u}]\neg A) \wedge & (3)\\ & \quad (A \rightarrow [(\mathtt{fchild} \cup \mathtt{right}); \mathbf{u}]\neg A)) & (4) \end{aligned}$$

Hence, using one such constraint for each identifier in $A \in \Sigma_{id}$, the ExpTime-completeness result in [21] gives us also a complexity characterization for our variant of *RXPath* root constraints that involve identifiers.

Theorem 11 ([21]). *Satisfiability of RXPath root constraints is* ExpTime*-complete.*

As mentioned, the technique for checking satisfiability of *RXPath* root constraints in [21] is based on a reduction to satisfiability in repeat-CDPDL, which so far has resisted implementation. Instead, by resorting to 2WATAs on finite trees as presented in Section 3, we offer a more promising path towards a viable implementation.

To make use of such a technique for checking satisfiability of *RXPath* root constraints, we need to modify the 2WATA $\mathbf{A}_\varphi$ in such a way that it accepts only binary trees that are well-formed (cf. Section 4). Indeed, a well-formed binary tree

[7] The hardness result holds also if all propositions are disjoint, and in the case where they represent standard XML document tags.

$T = (\Delta^T, \ell^T)$ induces a sibling tree $\pi_s(T)$. To define $\pi_s(T) = (\Delta^{T_s}, \cdot^{T_s})$, we define, by induction on Δ^T, a mapping π_s from Δ^T to words over $\mathbb{N}$ as follows:

- $\pi_s(\varepsilon) = \varepsilon$; if $hfc \in \ell^T(\varepsilon)$, then $\pi_s(1) = 1$;
- if $hfc \in \ell^T(x)$ and $\pi_s(x) = z{\cdot}n$, with $z \in \mathbb{N}^*$ and $n \in \mathbb{N}$, then $\pi_s(x{\cdot}1) = z{\cdot}n{\cdot}1$;
- if $hrs \in \ell^T(x)$ and $\pi_s(x) = z{\cdot}n$, with $z \in \mathbb{N}^*$ and $n \in \mathbb{N}$, then $\pi_s(x{\cdot}2) = z{\cdot}(n{+}1)$.

Then, we take Δ^{T_s} to be the range of π_s, and we define the interpretation function $\cdot^{T_s}$ as follows: for each $A \in \Sigma_a$, we define $A^{T_s} = \{\pi_s(x) \in \Delta^{T_s} \mid A \in \ell^T(x)\}$; similarly, for each $Id \in \Sigma_{id}$, we define $Id^{T_s} = \{\pi_s(x) \in \Delta^{T_s} \mid Id \in \ell^T(x)\}$. Notice that, since T is well-formed, Id^{T_s} contains at most one element. Note that the mapping π_s ignores irrelevant parts of the binary tree, e.g., if the label of a node x does not contain hfc, even if x has a 1-successor, such a node is not included in the sibling tree. Also, π_s can be considered the inverse of the mapping π_b defined in Section 4.

The 2WATA $\mathbf{A}^{wf}_{\varphi} = (\mathcal{L}, S, s_{ini}, \delta, \alpha)$, obtained by modifying $\mathbf{A}_{\varphi}$ so that it accepts (from ε) only trees that are well-formed, is defined as follows:

- The set of states is $S = S_{\varphi} \cup \{s_{ini}, s_{struc}\} \cup \{s_{Id}, n_{Id} \mid Id \in \Sigma_{id}\}$, where s_{ini} is the initial state, and the other additional states are used to check structural properties of well-formed trees.
- The transition function is constituted by all transitions in δ_{φ}, plus the following transitions ensuring that $\mathbf{A}^{wf}_{\varphi}$ accepts only well-formed trees.
  1. For each $\lambda \in \mathcal{L}$, there is a transition

     $$\delta(s_{ini}, \lambda) \;=\; (0, \varphi) \wedge (0, \neg ifc) \wedge (0, \neg irs) \wedge (0, \neg hrs) \wedge (0, s_{struc}) \wedge \textstyle\bigwedge_{Id \in \Sigma_{id}} (0, s_{Id})$$

     Such transitions (*i*) move to the initial state of $\mathbf{A}_{\varphi}$ to verify that φ holds at the root of the tree, (*ii*) check that the root of the tree is not labeled with ifc, irs or hrs, (*iii*) move to state s_{struc}, from which structural properties of the tree are verified, and (*iv*) move to states s_{Id} (for each $Id \in \Sigma_{id}$), from which the automaton verifies that a single occurrence of Id is present on the tree.
  2. For each $\lambda \in \mathcal{L}$, there is a transition

     $$\begin{aligned}\delta(s_{struc}, \lambda) \;=\; & ((0, \neg hfc) \vee ((1, ifc) \wedge (1, \neg irs) \wedge (1, s_{struc}))) \wedge \\ & ((0, \neg hrs) \vee ((2, irs) \wedge (2, \neg ifc) \wedge (2, s_{struc})))\end{aligned}$$

     Such transitions check that, (*i*) for each node labeled with hfc, its left child is labeled with ifc but not with irs, and (*ii*) for each node labeled with hrs, its right child is labeled with irs but not with ifc.
  3. For each $\lambda \in \mathcal{L}$ and each $Id \in \Sigma_{id}$ there are transitions

     $$\begin{aligned}\delta(s_{Id}, \lambda) \;=\; & ((0, Id) \wedge ((0, \neg hfc) \vee (1, n_{Id})) \wedge ((0, \neg hrs) \vee (2, n_{Id}))) \vee \\ & ((0, \neg Id) \wedge (0, hfc) \wedge (1, s_{Id}) \wedge ((0, \neg hrs) \vee (2, n_{Id}))) \vee \\ & ((0, \neg Id) \wedge (0, hrs) \wedge (2, s_{Id}) \wedge ((0, \neg hfc) \vee (1, n_{Id})) \\ \delta(n_{Id}, \lambda) \;=\; & (0, \neg Id) \wedge ((0, \neg hfc) \vee (1, n_{Id})) \wedge ((0, \neg hrs) \vee (2, n_{Id}))\end{aligned}$$

     Such transitions ensure that exactly one node of the tree is labeled with Id.
- The set of accepting states is $\alpha = \alpha_{\varphi}$. The states s_{ini} and s_{struc} form each a single element of the partition of states, where $\{s_{ini}\}$ precedes all other elements, and $\{s_{struc}\}$ follows them. The states s_{Id} and n_{Id} are added to the element of the partition containing all literals.

As for the size of $\mathbf{A}_{\varphi}^{wf}$, by Proposition 8, and considering that the additional states and transitions in $\mathbf{A}_{\varphi}^{wf}$ are constant in the size of φ, we get:

Proposition 12. *The number of states of $\mathbf{A}_{\varphi}^{wf}$ is linear in the size of φ.*

Theorem 13. *Let Γ be a (finite) set of RXPath root constraints, φ the conjunction of the constraints in Γ, and $\mathbf{A}_{\varphi}^{wf}$ the 2WATA constructed above. Then $\mathbf{A}_{\varphi}^{wf}$ is nonempty if and only if Γ is satisfiable.*

Theorem 14. *Checking the satisfiability of a (finite) set Γ of RXPath root constraints by checking nonemptiness of the 2WATA constructed above can be done in* EXPTIME.

6 Query Satisfiability and Query Containment

We deal now with query satisfiability and query containment under constraints, and we show that these problems can be reduced in linear time to satisfiability of *RXPath* root constraints. As a consequence, we get that these problems are EXPTIME-complete and that we can exploit for them the automata-based techniques developed in this paper.

In the following, we deal only with *RXPath* binary queries (i.e., path expressions), since *RXPath* unary queries (i.e., node expressions) can be rephrased as binary queries: indeed $\varphi^{T_s} = \{z \mid (z, z) \in (\varphi?)^{T_s}\}$.

We start our investigation with the query satisfiability problem. An *RXPath query Q is satisfiable* under a (finite) set of root constraints Γ if there exists a sibling tree T_s satisfying Γ such that Q^{T_s} is non-empty. Considering the semantics of *RXPath* queries and root constraints, it is immediate to verify that Q is satisfiable under Γ if and only if

$$\Gamma \cup \{\langle \mathbf{u}; Q \rangle \mathtt{true}\}$$

is satisfiable. Hence, query satisfiability under root constraints in *RXPath* can be linearly reduced to satisfiability of *RXPath* root constraints, and we get the following result

Theorem 15. *Query satisfiability under root constraints in RXPath is* EXPTIME*-complete.*

We now turn our attention to query containment under constraints, i.e., verifying whether for all databases satisfying a certain set of integrity constraints, the answer to a query is a subset of the answer to a second query. Checking containment of queries is crucial in several contexts, such as query optimization, query reformulation, knowledge-base verification, information integration, integrity checking, and cooperative answering. Obviously, query containment is also useful for checking equivalence of queries, i.e., verifying whether for all databases the answer to a query is the same as the answer to another query. For a summary of results on query containment in semistructured, see, e.g., [7].

Query containment under constraints in our setting is defined as follows: An *RXPath query Q_1 is contained in an RXPath query Q_2* under a set of *RXPath* constraints Γ, written $\Gamma \models Q_1 \subseteq Q_2$, if for every sibling tree T_s that satisfies all constraints in Γ,

we have that $Q_1^{T_s} \subseteq Q_2^{T_s}$. Again we can resort to root constraints satisfiability to verify containment. Namely: $\Gamma \models Q_1 \subseteq Q_2$ if and only if

$$\Gamma \cup \{\langle \mathbf{u}; Id_{st}?; Q_1; Id_{end}?\rangle \texttt{true}, [\mathbf{u}; Id_{st}?; Q_2; Id_{end}?]\texttt{false}\}$$

is unsatisfiable, where Id_{st} and Id_{end} are newly introduced identifiers.

We get that also query containment under root constraints in *RXPath* can be linearly reduced to unsatisfiability of *RXPath* root constraints.

Theorem 16. *Query containment under root constraints in RXPath is* EXPTIME*-complete.*

It follows that for the above problems of reasoning about queries under *RXPath* root constraints, we can exploit the automata-based techniques developed in this paper.

We conclude the section by observing that also view-based query answering has attracted the interest of the *XPath* community, e.g., [14]. It can be shown that we can adapt the above techniques based on a reduction to satisfiability of *RXPath* root constraints also to solve view-based query answering.

7 Conclusions

In this paper we have studied *RXPath*, a powerful mechanism for expressing structural queries and constraints in XML. We have presented symbolic automata-based techniques for evaluation of *RXPath* queries over XML trees, and for checking satisfiability of *RXPath* constraints, and we have illustrated how to apply the latter technique for both query containment and view-based query answering. Notably, the automata-theoretic techniques that we have introduced check for infinite computations on finite trees.

Acknowledgements This research has been partially supported by NSF grants CCR-0124077, CCR-0311326, CCF-0613889, ANI-0216467, and CCF-0728882.

References

1. L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. Marx, and M. de Rijke. PDL for ordered trees. *J. of Applied Non-Classical Logics*, 15(2):115–135, 2005.
2. G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: A practical study. In *Proc. of WebDB 2004*, pages 79–84, 2004.
3. M. Bojanczyk and P. Parys. XPath evaluation in linear time. In *Proc. of PODS 2008*, pages 241–250, 2008.
4. R. E. Bryant. Graph-based algorithms for Boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.
5. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
6. D. Calvanese, G. De Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *J. of Log. and Comp.*, 9(3):295–318, 1999.

7. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query answering and query containment over semistructured data. In G. Ghelli and G. Grahne, editors, *Revised Papers of the 8th International Workshop on Database Programming Languages (DBPL 2001)*, volume 2397 of *LNCS*, pages 40–61. Springer, 2002.
8. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.
9. J. Clark and S. DeRose. XML Path Language (XPath) version 1.0. W3C Recommendation, Nov. 1999. Available at `http://www.w3.org/TR/1999/REC-xpath-19991116`.
10. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at `http://www.grappa.univ-lille3.fr/tata/`, 2002.
11. S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs. In *Proc. of STOC'88*, pages 477–490, 1988.
12. J. E. Doner. Decidability of the weak second-order theory of two successors. *Notices Amer. Math. Soc.*, 12:819, 1965.
13. W. Fan. XML constraints: Specification, analysis, and applications. In *Proc. of DEXA 2005*, 2005.
14. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Rewriting regular XPath queries on XML views. In *Proc. of ICDE 2007*, pages 666–675, 2007.
15. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.
16. P. Genevès, N. Layaïda, and A. Schmitt. Efficient static analysis of XML paths and types. In *Proc. of the ACM SIGPLAN 2007 Conf. on Programming Language Design and Implementation (PLDI 2007)*, pages 342–351, 2007.
17. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. on Database Systems*, 30(2):444–491, 2005.
18. M. Jurdzinski. Small progress measures for solving parity games. In *Proc. of STACS 2000*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.
19. O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. of the ACM*, 47(2):312–360, 2000.
20. L. Libkin and C. Sirangelo. Reasoning about XML with temporal logics and automata. In *Proc. of LPAR 2008*, pages 97–112, 2008.
21. M. Marx. XPath with conditional axis relations. In *Proc. of EDBT 2004*, volume 2992 of *LNCS*, pages 477–494. Springer, 2004.
22. M. Marx. First order paths in ordered trees. In *Proc. of ICDT 2005*, volume 3363 of *LNCS*, pages 114–128. Springer, 2005.
23. S. Safra. On the complexity of ω-automata. In *Proc. of FOCS'88*, pages 319–327, 1988.
24. U. Sattler and M. Y. Vardi. The hybrid μ-calculus. In *Proc. of IJCAR 2001*, pages 76–91, 2001.
25. C. Schulte Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of Büchi automata. In *Proc. of the 10th Int. Conf. on the Implementation and Application of Automata*, 2005.
26. T. Schwentick. XPath query containment. *SIGMOD Record*, 33(1):101–109, 2004.
27. G. Slutzki. Alternating tree automata. *Theor. Comp. Sci.*, 41:305–318, 1985.
28. S. Tasiran, R. Hojati, and R. K. Brayton. Language containment using non-deterministic Omega-automata. In *Proc. of CHARME'95*, volume 987 of *LNCS*, pages 261–277. Springer, 1995.
29. B. ten Cate and L. Segoufin. XPath, transitive closure logic, and nested tree walking automata. In *Proc. of PODS 2008*, pages 251–260, 2008.
30. M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of ICALP'98*, volume 1443 of *LNCS*, pages 628–641. Springer, 1998.