Regular XPath: Constraints, Query Containment and View-Based Answering for XML Documents

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

[1] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
`calvanese@inf.unibz.it`

[2] Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Italy
`degiacomo,lenzerini@dis.uniroma1.it`

[3] Department of Computer Science, Rice University, Houston, TX, U.S.A.
`vardi@cs.rice.edu`

Abstract. In this paper we consider a powerful mechanism, called Regular *XPath*, for expressing queries and constraints over XML data, including DTDs and existential path constraints and their negation. Regular *XPath* extends XPath with binary relations over XML nodes specified by means two-way regular path queries. Our first contribution deals with checking satisfiability of Regular *XPath* constraints. While this problem could be reduced in terms of reasoning in repeat converse deterministic PDL, a well-known variant Propositional Dynamic Logic (PDL), the resulting technique would be of little practical use, due to the notorious difficulty of implementing efficient reasoners for such a logic. We therefore propose a direct algorithm for Regular *XPath* constraints satisfiability, based on checking emptiness of two way alternating automata on finite trees. We show how this algorithm can be implemented symbolically, by using Binary Decision Diagrams (BDDs) as the underlying data structure, which can be significantly more efficient than explicit graph-based algorithms. We then move to query containment and view based query answering for Regular *XPath*, and show that both problems can be reduced to checking satisfiability of Regular *XPath* constraints, thus allowing for taking advantage of the techniques developed for constraints satisfiability.

1 Introduction

XML[4] is becoming 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 mechanisms have been proposed for the specification of constraints that trees should satisfy in order to represent legal documents in a certain application domain. As pointed out in [9], such constraints can be classified into *structural* and *data value constraints*.

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

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

examples of formalisms allowing for expressing such constraints are DTDs (see footnote 1 and [5]), and XML Schema[5].

On the other hand, data-value constraints are used to enforce specific rules for the possible values associated with nodes, and/or to relate data values across the elements of the document. Popular constraints of this type are key and foreign-key constraints. Several recent papers carry out an extensive investigation of the decidability and complexity of reasoning about data-value constraints, also in the presence of some form of specification of structural constraints (e.g., [10]). A thorough analysis of semistructured and XML data-value constraints is reported in various surveys, see, e.g., [9].

This paper deals with structural constraints, by introducing a powerful mechanism, called Regular *XPath* (*RXPath*), for expressing both constraint of this type, and queries over XML trees. Our language stems from the work on *XPath* reported in [16, 17]. In particular, we extend *XPath* both with nominals, and with binary relations over XML nodes, expressed as two-way regular expressions over *XPath* axes. Nominals are a mechanism for denoting a single node in a document, and are similar to XML global identifiers built through the construct `ID`. While *XPath* queries traditionally select nodes from XML documents by specifying paths from the root, binary relations are abstractions for sets of node pairs connected by suitably specified paths. Notably, *RXPath* can express an array of popular structural constraints, including DTDs and existential path constraints, together with their negation. Additionally, the power of our language in expressing paths is the one of Propositional Dynamic Logic (PDL) [11] extended with converse, nominals, and deterministic programs. This combination of path-forming constructs results in one of the most expressive languages ever considered for specifying structural constraints in XML.

Our first contribution deals with checking satisfiability of *RXPath* constraints. We first mention that this problem could be reduced to reasoning in *Repeat-Converse-Deterministic PDL (repeat-CDPDL)* a well-known variant of PDL. Unfortunately, the reasoning technique resulting from this reduction is of little practical use by itself, due to the notorious difficulty of implementing efficient reasoners for such a logic [23, 24]. We address this issue by providing a direct algorithm for *RXPath* constraints satisfiability, based on checking emptiness of two-way alternating automata on finite trees [7]. The worst-case complexity of the algorithm is EXPTIME, and therefore matches the lower bound of the problem.

The second contribution of our work concerns the practical applicability of our technique. Towards this goal, we show how the automata-based algorithm can be implemented symbolically. The method makes use of Binary Decision Diagrams (BDDs) as the underlying data structure, and can be significantly more efficient than explicit graph-based algorithms [19].

The third contribution is a study of *RXPath* as a query language, with the goal of characterizing both query containment and view-based query answering for our language. We show that both problems can be reduced to checking satisfiability of *RXPath* constraints, thus enabling us to take advantage of the above-mentioned techniques de-

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

veloped for constraint satisfiability. Again, we argue that ours is the richest XML-based framework for which a solution to view-based query answering is provided.

2 Regular XPath

Following [17, 18], 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. Instead, 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`[6], though a node can have multiple identifiers in our case. Observe that in general sibling trees are more general than XML document 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. Formally, a *sibling tree* is a pair $T_s = (\Delta^{T_s}, \cdot^{T_s})$, where Δ^{T_s} is a tree (i.e., a complete prefix closed set of strings over $\mathbb{N}$), 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 [17, 18], 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) [11, 1], and adopt the PDL syntax to express 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 parenthesis for clarity):

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

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 `child`$^-$ and `right`$^-$, respectively. Also, we use the usual abbreviations `true` and `false`.

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 $\neg\langle P\rangle\neg\varphi$.

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

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

Fig. 1. Semantics of node and path expressions

In order to develop our techniques for inference on *RXPath*, we 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 (and hence ranked) trees.

We say that a path expression is *normalized* if it is expressed by making use of fchild and right only, if the $\cdot^-$ 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 expression 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 are used both to specify constraints and to express queries on XML documents. Queries are dealt with in Section 5, here we concentrate on constraints. An *RXPath root constraint* is a node expression intended to be true on the root of the document. The root constraint φ is *satisfied* in a sibling tree T_s if $\varepsilon \in \varphi^{T_s}$, i.e., the root is in the extension of φ in 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.

RXPath root constraints are indeed a quite powerful mechanism to describe structural properties of documents. As shown in [18], *RXPath* node expressions allow one to express all first-order definable sets of nodes, and this allows for quite sophisticated conditions as *RXPath* root constraints. Also they allow for capturing *DTDs*, *Specialized DTDs* [20] and the structural part of XML Schema Definitions.

In [17] it was shown that for *RXPath* root constraints without identifiers satisfiability is EXPTIME-complete, and this result carries over also to our variant of *RXPath*.

Theorem 1. *Satisfiability of RXPath root constraints is EXPTIME-complete.*

The known techniques for checking satisfiability of *RXPath* root constraints, as well as the EXPTIME-completeness result in [17], are based on a reduction to checking satisfiability in *Propositional Dynamic Logics (PDLs)*. Specifically, one can resort to *Repeat-Converse-Deterministic PDL (repeat-CDPDL)* [26], a variant of PDL that allows for expressing the finiteness of trees. While from a theoretical point of view this is fully satisfying, indeed satisfiability for repeat-CDPDL is EXPTIME-complete [27], from a practical point of view the decision procedures for repeat-CDPDL require the difficult determinization construction by Safra and parity games, which up to now have resisted implementations that work efficiently [23, 24]. So far, no implementation of repeat-CDPDL has been announced. Thus, the above complexity result does not induce viable automated reasoning techniques for reasoning for *RXPath* constraints. Hence, next, we look at a direct reasoning technique based on automata on finite (as opposed to infinite) trees. As we argue later, this offer a more promising path towards a viable reasoning technique.

3 Satisfiability of root constraints via automata

We work on complete 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 the first child* (resp., *has the right sibling*).

Formally, we consider binary trees whose nodes are labeled with subsets of $\Sigma \cup \{\mathit{ifc}, \mathit{irs}, \mathit{hfc}, \mathit{hrs}\}$. We call 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 $\mathit{Id} \in \Sigma_{id}$, there is at most one node x of T with $\mathit{Id} \in \ell^T(x)$.

A well-formed binary tree $T = (\Delta^T, \ell^T)$, induces a sibling tree $T_s = (\Delta^{T_s}, \cdot^{T_s})$ defined as follows. We inductively define on Δ^T both Δ^{T_s} and a mapping π as follows:

- $\pi(\varepsilon) = \varepsilon$, and $\varepsilon \in \Delta^{T_s}$;
- if $\ell^T(\varepsilon)$ contains hfc, then $1 \in \Delta^{T_s}$ and $\pi(1) = 1$;
- if $\ell^T(x)$ contains hfc and $\pi(x) = z{\cdot}n$, then $z{\cdot}n{\cdot}1 \in \Delta^{T_s}$ and $\pi(x{\cdot}1) = z{\cdot}n{\cdot}1$;
- if $\ell^T(x)$ contains hrs and $\pi(x) = z{\cdot}n$, then $z{\cdot}(n{+}1) \in \Delta^{T_s}$ and $\pi(x{\cdot}2) = z{\cdot}(n{+}1)$.

Then, we define the interpretation function $\cdot^{T_s}$ as follows: for each $A \in \Sigma_a$, we define $A^{T_s} = \{\pi(x) \in \Delta^{T_s} \mid A \in \ell^T(x)\}$; similarly, for each $\mathit{Id} \in \Sigma_{id}$, we define $\mathit{Id}^{T_s} = \{\pi(x) \in \Delta^{T_s} \mid \mathit{Id} \in \ell^T(x)\}$. Notice that, since T is well-formed, Id^{T_s} contains at most one element.

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. Given a (finite) set of *RXPath* root constraints Γ, let φ be the normalized node expression formed by the conjunction of the constraints in Γ. We can construct a 2ATA $\mathbf{A}_\varphi$, with a number of states linear in the size of φ, such that, for each well-formed tree T accepted by $\mathbf{A}_\varphi$, φ is satisfied in the sibling tree induced by T.

Lemma 1. *Let Γ be a (finite) set of RXPath root constraints, φ the normalized node expression formed by the conjunction of the constraints in Γ, $\mathbf{A}_\varphi$ the 2ATA constructed from φ, T a well-formed tree, and T_s the sibling tree induced by T. Then $\mathbf{A}_\varphi$ accepts T if and only if Γ is satisfied in T_s.*

Next we can modify $\mathbf{A}_\varphi$ so as to consider only trees that are well-formed. The resulting 2ATA is denoted $\mathbf{A}_\varphi^{wf}$, still with a number of states linear in the size of φ.

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

Theorem 3. *Checking the satisfiability of a (finite) set Γ of RXPath root constraints by checking nonemptiness of the 2ATA constructed above can be done in EXPTIME.*

4 Symbolic Algorithms

As we showed earlier, satisfiability of *RXPath* root constraints can be reduced to satisfiability of formulas in repeat-CDPDL, which is in EXPTIME. This upper bound, however, is established using sophisticated infinite-tree automata-theoretic techniques (cf., e.g., [22]), which so far have resisted attempts at practically efficient implementation, due to the use of Safra's determinization construction [21] and parity games [13]. 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 [7], nonemptiness for 2ATA can be tested in time that is exponential in the number of states and linear in the size of the alphabet. This result is just sketched in [7]. We provide more details here, based on more recent developments in [27] for infinite-tree automata, in order to show that our 2ATA-based decision procedure can be implemented using a *symbolic* approach, which has the potential to be capable of handling automata with large states spaces [4]. The restriction to finite trees enables us to simplify the framework of [27] significantly.

The first step in testing nonemptiness of 2ATA is converting them to equivalent one-way nondeterministic tree automata (NTA). A 2ATA $\mathbf{A} = (\mathcal{L}, S, s_0, \delta)$ is an NTA if the directions -1 and 0 are not used in δ and, 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). (For simplicity we focus here on binary trees.) In other words, each disjunct corresponds to sending at most one "subprocess" in each direction. While for 2ATA we have separate input tree and run tree, for NTA 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, but a different labeling.

The advantage in working with NTA is that they have a very simple, linear-time, nonemptiness algorithm. The algorithm computes, in a bottom-up fashion the set Acc of states of the automaton that can lead to acceptance. Then, all that is left to check is that s_0 is one of these states. Initially, Acc is empty. In each iteration, we add to Acc all states s such that there is a letter a where $\delta(s, a)$ is satisfied by some assignment that assigns true to the atoms $(1, s_1)$ and $(1, s_2)$, for some $s_1, s_2 \in Acc$. Using techniques used in checking satisfiability of propositional Horn formulae [8], nonemptiness testing can be done in time that is linear in the size of $\mathbf{A}$.

The linear-time algorithm for NTA nonemptiness requires an explicit enumeration of all automaton states. When the number of states is extremely large, this approach may be infeasible. Instead we may want to take advantage of the fact that large sets of states may be described compactly using an appropriate symbolic representation. A representation that has enjoyed significant success in the context of automated verification [4] is that of *binary decision diagrams* [3]. A binary decision diagram (BDD) is a rooted directed acyclic graph that has only two terminal nodes labeled 0 and 1. Every non-terminal node is labeled with a Boolean variable and has two outgoing edges labeled 0 and 1. An ordered binary decision diagram (OBDD) is a BDD with the constraint that the input variables are ordered and every path in the OBDD visits the variables in ascending order. An ROBDD is an OBDD where every node represents a distinct logic function. Experience has shown that in many cases Boolean functions with very large sets of support can be described by ROBDDs very compactly. CUDD[7] is a software package that provides functions for the manipulation of Boolean functions, based on the reduced, ordered, binary decision diagram (ROBDD) representation. In particular, CUDD provides APIs for set-theoretic operations such as union and complement, for existential quantification, and for equality checking between two functions. To implement the NTA-nonemptiness algorithm symbolically we need to be able to describe sets of automaton states symbolically. The basic iteration of the algorithm can then be implemented in terms of ROBDD operations. The whole algorithm can be implemented symbolically without ever constructing the NTA explicitly. For examples of this approach for K and CTL, see [19, 15].

We believe that a symbolic approach offers a viable path towards an automated reasoning technique for *RXPath*.

It remains to describe the translation of 2ATAs to NTAs, showing that sets of NTA's states lend themselves to symbolic representation. Given a 2ATA $\mathbf{A}$ and an input tree T as above, a *strategy for* $\mathbf{A}$ *on* T is a mapping $\tau : \Delta^T \rightarrow 2^{S\times[k]\times S}$. Thus, each label in a strategy is an edge-$[k]$-labeled directed graph on S. Intuitively, each label is a set of transitions. For each label ζ, 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 $c_i \in [k]$ 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. We say that τ is *accepting* if it has no infinite paths.

Proposition 1. *A 2ATA* $\mathbf{A}$ *accepts an input tree* T *iff* $\mathbf{A}$ *has an accepting strategy tree for* T.

[7] `http://vlsi.colorado.edu/~fabio/CUDD/`

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 $\eta : \Delta^T \to 2^{S \times S}$. Thus, each label in an annotation is a directed graph on 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. The conditions are: (a) if $(s, s') \in \eta(x)$ and $(s', s'') \in \eta(x)$, then $(s, s'') \in \eta(x)$, (b) if $(s, 0, s') \in \tau(x)$ then $(s, s') \in \eta(x)$, (c) if $x = y{\cdot}i$, $(s, -1, s') \in \tau(x)$, $(s', s'') \in \eta(y)$, and $(s'', i, s''') \in \tau(x)$, then $(s, s''') \in \eta(x)$, (d) if $y = x{\cdot}i$, $(s, i, s') \in \tau(x)$, $(s', s'') \in \eta(y)$, and $(s'', -1, s''') \in \tau(y)$, then $(s, s''') \in \eta(x)$. The annotation η is accepting if for no node $x \in \Delta^T$ and state $s \in S$ we have that $(s, s) \in \eta(x)$. In other words, η is accepting if it contains no cycles.

Proposition 2. *A 2ATA* $\mathbf{A}$ *accepts an input tree* T *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 4. *Let* $\mathbf{A}$ *be a 2ATA. Then there is an NTA* $\mathbf{A}_n$ *such that* $\mathcal{L}(\mathbf{A}) = \mathcal{L}(\mathbf{A}_n)$. *The number of states in* $\mathbf{A}_n$ *is exponential in the number of states of* $\mathbf{A}$.

Proof (sketch). Let $\mathbf{A} = (\mathcal{L}, S, s_0, \delta)$ and let the input tree be $T = (\Delta^T, \ell^T)$. The automaton $\mathbf{A}_n$ guesses mappings $\tau : \Delta^T \to S \times [k] \times S$ and $\eta : \Delta^T \to S \times S$ and checks that τ is a strategy for $\mathbf{A}$ on T and η is an accepting annotation for $\mathbf{A}$ on T with respect to τ. The state space of $\mathbf{A}_n$ is $2^S \times 2^{S2}$; intuitively, after reading the label of a node x, $\mathbf{A}_n$ needs to remember the value of $\mathit{state}(\tau(x))$ and $\eta(x)$. The transition function of $\mathbf{A}_n$ checks that τ and η satisfies all the required conditions.

The key feature of the state space of $\mathbf{A}_n$ is the fact that states are pairs consisting of subsets of S and $S2$. Thus, a set of states of $\mathbf{A}_n$ can be described by a Boolean function on the domain $S3$. Similarly, the transition function of $\mathbf{A}_n$ can also be described as a Boolean function. Such functions can be represented by ROBDDs, enabling a symbolic approach to nonemptiness testing of 2ATAs. We note that the framework of [27] also converts a 2ATA (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.

5 Satisfiability, containment and view-based answering for RXPath queries

In this section we consider *RXPath* as query language, and consider a number of problems involving *RXPath* queries. An *RXPath query* Q is an *RXPath* path expression that, when evaluated over a sibling tree T_s, returns the set of pairs of nodes Q^{T_s}. We do not introduce explicitly *RXPath* node expressions as (unary) queries, since they can be rephrased as path expressions: indeed $\varphi^{T_s} = \{z \mid (z,z) \in (\varphi?)^{T_s}\}$.

Besides the basic task of query answering, i.e., evaluating a query over a database, data and knowledge representation systems should support other reasoning services related to querying. In particular, we are interested in query satisfiability, query containment under constraints, and view-based query answering. For each of such problem we show a linear time reduction to satisfiability of *RXPath* root constraints. As a consequence, we get that all such problems are EXPTIME-complete. Moreover the reductions allow us to exploit the automata-based techniques developed in the previous sections to deal with such problems as well.

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, where $\mathbf{u}$ is an abbreviation for $(\mathtt{fchild} \cup \mathtt{right})^*$. Hence we get:

Proposition 3. *Query satisfiability under root constraints in RXPath can be linearly reduced to satisfiability of RXPath root constraints.*

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, which is crucial in several contexts.

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

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

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

Proposition 4. *Query containment under root constraints in RXPath can be linearly reduced to unsatisfiability of RXPath root constraints.*

View-based query processing is another form of reasoning that has recently drawn a great deal of attention in the database community [12]. In several contexts, such as data integration, query optimization, query answering with incomplete information, and data warehousing, the problem arises of processing queries posed over the schema of a virtual database, based on a set of materialized views, rather than on the raw data in the database [25, 14]. integration system exports a global virtual schema over which user queries are posed, and such queries are answered based on the data stored in a collection of data sources, whose content in turn is described in terms of views over the global schema. In such a setting, each data source corresponds to a materialized view, and the global schema exported to the user corresponds to the schema of the virtual database. Notice that typically, in data integration, the data in the sources are correct (i.e., sound) but incomplete with respect to their specification in terms of the global schema. This is due the fact that typically the global schema is not designed taking the sources into account, but rather the information needs of users. Hence it may not be possible to precisely describe the information content of the sources. In this paper we will concentrate on this case (*sound views*)

Consider now a document that is accessible only through a collection of views expressed as *RXPath* queries, and suppose we need to answer an *RXPath* query over the document only on the basis of our knowledge on the views. Specifically, the collection of views is represented by a finite set $\mathcal{V}$ of *view symbols*, each denoting a binary relation. Each view symbol $V \in \mathcal{V}$ has an associated *view definition* Q_V and a *view extension* $\mathcal{E}_V$. The view definition Q_V is simply an *RXPath* query. The view extension $\mathcal{E}_V$ is a set of pairs of *node references*, where each node reference is either an identifier, or an explicit path expression that is formed only by chaining `fchild` and `right` and that identifies the node by specifying how to reach it from the root. Observe that a node reference a is interpreted in a sibling tree T_s as a singleton set of nodes a^{T_s}. We use $(\mathcal{E}_V)^{T_s}$ to denote the set of pairs of nodes resulting from interpreting the node references in T_s. We say that a *sibling tree T_s satisfies a view V* if $(\mathcal{E}_V)^{T_s} \subseteq (Q_V)^{T_s}$. In other words, in T_s all the tuples denoted by $(\mathcal{E}_V)^{T_s}$ must appear in $(Q_V)^{T_s}$, but $(Q_V)^{T_s}$ may contain tuples not in $(\mathcal{E}_V)^{T_s}$.

Given a set $\mathcal{V}$ of views, and an *RXPath* Q, the set of *certain answers* to Q with respect to $\mathcal{V}$ under root constraints Γ is the set $cert_{Q,\mathcal{V},\Gamma}$ of pairs (a, b) of node references such that $(a^{T_s}, b^{T_s}) \in Q^{T_s}$ for every sibling tree T_s satisfying each $V \in \mathcal{V}$ and each constraint in Γ. *View-based query answering* under root constraints consists in deciding whether a given pair of node references is a certain answer to Q with respect to $\mathcal{V}$.

Also view-based query answering can be reduced to satisfiability of root constraints. Given a view V, with extension $\mathcal{E}_V$ and definition Q_V, for each $(a, b) \in \mathcal{E}_V$:

- if both a and b are identifiers, denoted by Id_a and Id_b respectively, then we introduce the root constraint $\langle \mathbf{u}; Id_a?; Q_V; Id_b?\rangle\mathtt{true}$;
- if both are a and b are explicit path expressions, denoted by P_a and P_b respectively, then we introduce the root constraint $\langle P_a; Q_V; P_b^-; Id_\varepsilon?\rangle\mathtt{true}$, where $Id_\varepsilon = [\mathtt{fchild}^- \cup \mathtt{right}^-]\mathtt{false}$ expresses that the final node reached by the path is the root, i.e., the only node that has no parent node;
- the other two cases are an obvious combination of the above.

Let $\Gamma_{\mathcal{V}}$ be the set of root *RXPath* constraints corresponding to the set of *RXPath* views $\mathcal{V}$, Q an *RXPath* query and Γ a finite set of root constraints. Then a pair (a, b) of node references belongs to $cert_{Q,\mathcal{V},\Gamma}$ if and only if the following set of root constraints is unsatisfiable:

- $\Gamma \cup \Gamma_{\mathcal{V}} \cup \{[\mathbf{u}; Id_a?; Q; Id_b?]\mathtt{false}\}$, if both a and b are identifiers, denoted by Id_a and Id_b respectively;
- $\Gamma \cup \Gamma_{\mathcal{V}} \cup \{[P_a; Q; P_b^-; Id_\varepsilon?]\mathtt{false}\}$ where again $Id_\varepsilon = [\mathtt{fchild}^- \cup \mathtt{right}^-]\mathtt{false}$, if both a and b are explicit path expressions, denoted by P_a and P_b respectively;
- similarly for the other two cases.

Hence we can state:

Proposition 5. *View based query answering under root constraints in RXPath can be linearly reduced to unsatisfiability of RXPath root constraints.*

6 Conclusions

In this paper we have studied *RXPath*, a powerful mechanism for expressing structural constraints and queries in XML. We have presented symbolic automata-based techniques for checking satisfiability of *RXPath* constraints, and we have illustrated how to apply these techniques for both the query containment and the view-based query answering problem. Of course, further algorithmic-engineering work is needed to demonstrate the practicality of our proposed approach. (See [19] for such work in the context of the modal logic K, where issues such as state-space representation, symbolic variable order, and formula-rewriting techniques are discussed.)

There are a number of interesting open problems related to *RXPath*, and they will be the subject of further research work. First, we aim at exploring extensions of the constraint language studied in the paper. One extension is with the so-called graded modalities, similar to number restrictions in Description Logics [2, 5], allowing one to impose conditions on the number of nodes that are reachable from a source node by means of a specified axis, and satisfy a given additional property. We conjecture that our satisfiability algorithm can be directly extended to deal with this type of constraints. A more complex class of constraints are those extending graded modalities so as to impose conditions on the number of nodes that are reachable from a given node by means of a specified path. As for queries, we aim at moving to interesting fragments of XQuery. The first fragment we aim at addressing corresponds to the language of conjunctive *RXPath*, whose queries are formed as conjunctions of atoms, where each atom denotes a binary relation expressed in *RXPath*. Another direction of inquiry is identifying the data complexity of view-based query processing of *RXPath* queries, which might be lower than the combined complexity studied here (cf. [6]). It is an open question whether the constraint-theoretic techniques used in [6] to obtained improved bounds for data complexity for graph-like structures can be adapted to the tree-like structures of XML data.

Acknowledgments

This research has been partially supported by the EU FP6 FET project TONES (Thinking ONtologiES) under contract number FP6-7603.

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. F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 2, pages 43–95. Cambridge University Press, 2003.
3. R. E. Bryant. Graph-based algorithms for Boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.

4. 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.
5. 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.
6. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.
7. 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.
8. W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. of Logic Programming*, 1(3):267–284, 1984.
9. W. Fan. XML constraints: Specification, analysis, and applications. In *Proc. of DEXA 2005*, 2005.
10. W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. of the ACM*, 49(3):368–406, 2002.
11. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.
12. A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.
13. M. Jurdzinski. Small progress measures for solving parity games. In *Proc. of STACS 2000*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.
14. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.
15. W. Marrero. Using BDDs to decide CTL. In *Proc. of the 11th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005)*, volume 3440 of *LNCS*, pages 222–236. Springer, 2005.
16. M. Marx. Conditional XPath, the first order complete XPath dialect. In *Proc. of PODS 2004*, pages 13–22, 2004.
17. M. Marx. XPath with conditional axis relations. In *Proc. of EDBT 2004*, volume 2992 of *LNCS*, pages 477–494. Springer, 2004.
18. M. Marx. First order paths in ordered trees. In *Proc. of ICDT 2005*, volume 3363 of *LNCS*, pages 114–128. Springer, 2005.
19. G. Pan, U. Sattler, and M. Y. Vardi. BDD-based decision procedures for K. In *Proc. of CADE 2002*, pages 16–30, 2002.
20. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. of PODS 2000*, pages 35–46, 2000.
21. S. Safra. On the complexity of ω-automata. In *Proc. of FOCS'88*, pages 319–327, 1988.
22. U. Sattler and M. Y. Vardi. The hybrid μ-calculus. In *Proc. of IJCAR 2001*, pages 76–91, 2001.
23. 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.
24. 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.
25. J. D. Ullman. Information integration using logical views. In *Proc. of ICDT'97*, volume 1186 of *LNCS*, pages 19–40. Springer, 1997.
26. M. Y. Vardi. The taming of converse: Reasoning about two-way computations. In R. Parikh, editor, *Proc. of the 4th Workshop on Logics of Programs*, volume 193 of *LNCS*, pages 413–424. Springer, 1985.
27. 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.