Reasoning about XML with Temporal Logics and Automata

Leonid Libkin[1] and **Cristina Sirangelo**[1,2]

[1] University of Edinburgh
[2] LSV, ENS–Cachan, INRIA

Abstract. We show that problems arising in static analysis of XML specifications and transformations can be dealt with using techniques similar to those developed for static analysis of programs. Many properties of interest in the XML context are related to navigation, and can be formulated in temporal logics for trees. We choose a logic that admits a simple single-exponential translation into unranked tree automata, in the spirit of the classical LTL-to-Buchi automata translation. Automata arising from this translation have a number of additional properties; in particular, they are convenient for reasoning about unary node-selecting queries, which are important in the XML context. We give two applications of such reasoning: one deals with a classical XML problem of reasoning about navigation in the presence of schemas, and the other relates to verifying security properties of XML views.
Keywords Query automata, static analysis, temporal logics, XML

1 Introduction

Static analysis of XML specifications and transformations has been the focus of many recent papers (see, e.g., [1, 4, 6, 8, 10, 13, 16, 24, 25, 35]). Typical examples include consistency of type declarations and constraints, or of schema specifications and navigational properties, or containment of XPath expressions. They found application in query optimization, access control, data exchange, and reasoning about security properties of views, among others.

Many XML specifications – for example, various schema formalisms – are automata-based. Furthermore, there is a close connection between XML navigation, which is a key component of query languages, and temporal logics used in the field of verification [5, 26, 25, 22, 16]. Thus, it is very natural to adapt automata-based techniques developed by the verification community (cf. [11]) for XML static analysis problems involving schemas and navigation.

Examples of such usage exist, but by and large they take existing verification tools, and attempt to reshape the problem at hand so that those tools would be applicable to it. For example, [25] shows how to reason about XML navigation language XPath and XML schemas by encoding them in PDL. While it achieves provably optimal EXPTIME-bound, it does so by a rather complicated algorithm (for example, it uses, as a black box for one of its steps, a translation

from PDL into a certain type of tree automata [40], for which no efficient implementations exist). Another example of such reasoning [16] goes via much better implementable μ-calculus, but the technique only guarantees $n^{O(n)}$ algorithms for problems for which $2^{O(n)}$ algorithms exist.

We propose an alternative approach: instead of using verification techniques as-is in the XML context, we adapt them to get better static analysis algorithms. The present paper can be viewed as a proof-of-concept paper: we demonstrate one logic-to-automata translation targeted to XML applications, which closely resembles the standard Vardi-Wolper's LTL-to-Büchi translation [39], and show that it is easily applicable in two typical XML reasoning tasks.

Typically, temporal logic formulae are translated into either nondeterministic or alternating automata; for LTL, both are possible [39, 37]. We believe that both should be explored in the XML context. For this paper, we concentrate on the former. A recent workshop paper [10] developed an alternating-automata based approach; it handled a more expressive navigation language, but did not work out connections with XML schemas, as we do here.

Our goal is to find a clean direct translation from a logical formalism suitable for expressing many XML reasoning tasks, into an automata model. Towards that end, we use a simple LTL-like logic for trees, which we call $\mathrm{TL}^{\mathrm{tree}}$, rather than a W3C-designed language (but we shall show that such languages can be easily translated into $\mathrm{TL}^{\mathrm{tree}}$). This logic was defined in [34], and it was recently used in the work on XPath extensions [26], and as a key ingredient for an expressively-complete logic for reasoning about procedural programs [2, 3].

The translation will produce a bit more than automata rejecting or accepting trees; instead it will produce *query automata* [30, 28, 15] which can also select nodes from trees in their successful runs. The ability to produce such automata is not surprising at all (since in the Vardi-Wolper construction states are sets of formulae and successful runs tell us which formulae hold in which positions). Nonetheless, it is a very useful feature for XML reasoning, since many XML data processing tasks are about *node-selecting queries* [18, 30, 36, 29]. Furthermore, additional properties of query automata arising in the translation make operations such as complementation and testing containment very easy for them. Consequently, it becomes easier to combine several reasoning tasks.

Organization In Section 2 we give examples of XML reasoning where the logic/automata connection would be useful. Section 3 describes unranked trees and automata for them. In Section 4 we present the logic $\mathrm{TL}^{\mathrm{tree}}$ and various XPath formalisms, and give an easy translation of XPath into $\mathrm{TL}^{\mathrm{tree}}$. In Section 5 we give a translation from $\mathrm{TL}^{\mathrm{tree}}$ to query automata. Section 6 applies this translation in complex reasoning tasks involving schemas and navigation in XML documents, and Section 7 gives an application to reasoning about XML views.

2 Motivating examples

We now consider two examples of XML static analysis problems that will later be handled by restating these problems with the help of $\mathrm{TL}^{\mathrm{tree}}$ and the automata

translation. While formal definitions will be given later, for the reader not fluent in XML the following abstractions will be sufficient. First, XML documents themselves are labeled unranked trees (that is, different nodes can have a different number of children). XML schemas describe how documents are structured; they may be given by several formalisms that are all subsumed by tree automata. The most common of such formalisms is referred to as DTDs (document type definitions). And finally XPath is a navigational language; an XPath expression for now can be thought of as selecting a set of nodes in a tree.

Reasoning about schemas and navigation A common static analysis problem in the XML context, arising in query optimization and consistency checking, is the interaction of navigational properties (expressed, for example, in XPath) with schemas (often given as DTDs). Known results about the complexity of problems such as XPath containment [35], or XPath/DTD consistency [6], are typically stated in terms of completeness for various intractable complexity classes. They imply unavoidability of exponential-time algorithms, but they do not necessarily lead to reasonable algorithms that can be used in practice.

To illustrate this, consider the *containment problem* of XPath expressions under a DTD, i.e., checking whether for all trees satisfying a DTD d, the set of nodes selected by e_1 is contained in the set selected by e_2 (written as $d \models e_1 \subseteq e_2$). Automata-based algorithms would either translate XPath directly into automata (which could depend heavily on a particular syntactic class [31]), or attempt a generic translation via an existing logic. The second approach, taken by [25, 6, 16], translates e_1, e_2, and d into formulae of expressive logics such as PDL (in [25]) or μ-calculus (in [16]). Then one uses techniques of [40, 38] to check if there exists a finite tree T satisfying d and a node s in T witnessing $e_1(s) \wedge \neg e_2(s)$, i.e., a counterexample to the containment. PDL and μ-calculus have been chosen because of their ability to encode XML schemas, e.g., DTDs, but, as we shall see, this is easy to avoid.

While this is very much in the spirit of the traditional logic/automata connection used so extensively in static analysis of programs, there are some problems with this approach as currently used. The logics used were chosen because of their ability to encode DTDs, but this makes the constructions apply several algorithms as black-boxes. For example, the PDL construction of [25] combines a translation into PDL with converse on binary trees, a rather complex automata model of [40] together with an extra automaton that restricts it to finite trees. Second, we do not get a concise description of the set of all possible counterexamples, rather a yes or no answer. And third, the high expressiveness of logics comes at a cost. The running time of algorithms that go via μ-calculus is $n^{O(n)}$ [16]. For the PDL approach [25], the running time is $2^{O(\|e_1\|+\|e_2\|+\|d\|)}$, where $\|\cdot\|$ denotes the size. In several applications, we would rather avoid the $2^{O(\|d\|)}$ factor, since many DTDs are computer-generated from database schemas and could be very large, while XPath expressions tend to be small.

The translation we propose is a direct and simple construction, and does not rely on complicated algorithms such as the PDL-to-automata translation.

It produces a concise description of *all* possible counterexamples, which can be reused later. Finally, it exhibits an exponential blowup in the size of e_1 and e_2, but remains polynomial in the size of the schema.

Reasoning about views and query answers Often the user sees not a whole XML document, but just a portion of it, V (called a *view*), generated by a query. Such a query typically specifies a set of nodes selected from a source document, and thus can be represented by a query automaton $\mathcal{QA}_\mathcal{V}$: i.e., an extension of a tree automaton that can select nodes in trees; a formal definition will be given shortly.

If we only have access to V, we do not know the source document that produced it, as there could be many trees T satisfying $V = \mathcal{QA}_\mathcal{V}(T)$. We may know, however, that every such source has to satisfy some schema requirements, presented by a tree automaton $\mathcal{A}$. A common problem is to check whether V may reveal some information about the source. If $\mathfrak{Q}$ is a Boolean (yes/no) query, one defines the *certain answer* to $\mathfrak{Q}$ over V to be true iff $\mathfrak{Q}$ is true in every possible T that generates V:

$$\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V) \quad = \quad \bigwedge\{\mathfrak{Q}(T) \;\mid\; V = \mathcal{QA}_\mathcal{V}(T), \;\; T \text{ is accepted by } \mathcal{A}\}$$

Now if by looking at V, we can conclude that $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V)$ is true, then V reveals that $\mathfrak{Q}$ is true in an unknown source. If $\mathfrak{Q}$ is a containment statement $e_1 \subseteq e_2$, such an inclusion could be information that needs to be kept secret (e.g., it may relate two different groups of people). For more on this type of applications, see [13, 14].

Suppose $\mathfrak{Q}$ itself is definable by an automaton $\mathcal{A}_\mathfrak{Q}$. If we can convert automata $\mathcal{A}_\mathfrak{Q}$, $\mathcal{A}$, and the query automaton $\mathcal{QA}_\mathcal{V}$ into a new automaton $\mathcal{A}^*$ that accepts V iff $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V)$ is false, then acceptance by $\mathcal{A}^*$ gives us some assurances that the secret is not revealed. Furthermore, since views are often given by XPath expressions, and e_1 and e_2 are often XPath expressions too, an efficient algorithm for constructing $\mathcal{A}^*$ would give us a verification algorithm exponential in (typically short) XPath expressions defining e_1, e_2, and $\mathcal{V}$, and polynomial in a (potentially large) expression defining the schema.

In fact, we shall present a polynomial-time construction for $\mathcal{A}^*$ for the case of subtree- (or upward-closed) queries [7]. In that case, combining it with previous efficient translations from logical formulae into query automata, we get efficient algorithms for verifying properties of views.

3 Unranked trees and automata

Unranked trees XML documents are normally abstracted as labeled unranked trees. We now recall the standard definitions, see [29, 22, 36]. Nodes in unranked trees are elements of $\mathbb{N}^*$, i.e. strings of natural numbers. We write $s \cdot s'$ for the concatenation of strings, and ε for the empty string. The basic binary relations on $\mathbb{N}^*$ are the *child relation*: $s \prec_{\text{ch}} s'$ if $s' = s \cdot i$, for some $i \in \mathbb{N}$, and the

next-sibling relation: $s' \prec_{\text{ns}} s''$ if $s' = s \cdot i$ and $s'' = s \cdot (i+1)$ for some $s \in \mathbb{N}^*$ and $i \in \mathbb{N}$. The *descendant* relation $\prec^*_{\text{ch}}$ and the younger sibling relation $\prec^*_{\text{ns}}$ are the reflexive-transitive closures of $\prec_{\text{ch}}$ and $\prec_{\text{ns}}$.

An *unranked tree domain* D is a finite prefix-closed subset of $\mathbb{N}^*$ such that $s \cdot i \in D$ implies $s \cdot j \in D$ for all $j < i$. If Σ is a finite alphabet, an *unranked tree* is a pair $T = (D, \lambda)$, where D is a tree domain and λ is a labeling function $\lambda : D \to \Sigma$.

Unranked tree automata and XML schemas A *nondeterministic unranked tree automaton* (cf. [29, 36]) over Σ-labeled trees is a triple $\mathcal{A} = (Q, F, \delta)$ where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and δ is a mapping $Q \times \Sigma \to 2^{Q^*}$ such that each $\delta(q, a)$ is a regular language over Q. We assume that each $\delta(q, a)$ is given as an NFA. A *run* of $\mathcal{A}$ on a tree $T = (D, \lambda)$ is a function $\rho_{\mathcal{A}} : D \to Q$ such that if $s \in D$ is a node with n children, and $\lambda(s) = a$, then the string $\rho_{\mathcal{A}}(s \cdot 0) \cdots \rho_{\mathcal{A}}(s \cdot (n-1))$ is in $\delta(\rho_{\mathcal{A}}(s), a)$. Thus, if s is a leaf labeled a, then $\rho_{\mathcal{A}}(s) = q$ implies that $\varepsilon \in \delta(q, a)$. A run is *accepting* if $\rho_{\mathcal{A}}(\varepsilon) \in F$, and a tree is *accepted* by $\mathcal{A}$ if an accepting run exists. Sets of trees accepted by automata $\mathcal{A}$ are called regular and denoted by $L(\mathcal{A})$.

There are multiple notions of *schemas* for XML documents, DTDs being the most popular one. What is common for them is that they are subsumed by the power of unranked tree automata, and each specific formalism has a simple (often linear time) translation into an automaton [36]. So when we speak of XML schemas, we shall assume that they are given by unranked tree automata.

Query automata It is well known that automata capture the expressiveness of MSO sentences over finite and infinite strings and trees. The model of query automata [30] captures the expressiveness of MSO formulae $\varphi(x)$ with one free first-order variable – that is, MSO-definable unary queries. We present here a nondeterministic version, as in [28, 15].

A *query automaton (QA)* for Σ-labeled unranked trees is a tuple $\mathcal{QA} = (Q, F, Q_s, \delta)$, where (Q, F, δ) is an unranked tree automaton, and $Q_s \subseteq Q$ is the set of *selecting states*. Each run ρ of $\mathcal{QA}$ on a tree $T = (D, \lambda)$ defines the set $S_\rho(T) = \{s \in D \mid \rho(s) \in Q_s\}$ of nodes assigned a selecting state. The unary query defined by $\mathcal{QA}$ is then, under the *existential semantics*,

$$\mathcal{QA}^{\exists}(T) \;=\; \bigcup \{S_\rho(T) \mid \rho \text{ is an accepting run of } \mathcal{QA} \text{ on } T\}.$$

Dually, one can define $\mathcal{QA}^{\forall}(T)$ under the *universal semantics* as the intersection of $S_\rho(T)$'s. Both semantics capture the class of unary MSO queries [28].

These notions are not very convenient for reasoning tasks, as many runs need to be taken into account – different nodes may be selected in different runs. Also, it makes operations on query automata hard computationally: for example, a natural notion of complement for an existential-semantics QA will be expressed as a universal semantics QA, requiring an exponential time algorithm to convert it back into an existential QA.

To remedy this, we define a notion of *single-run query automata* as QAs (Q, F, Q_s, δ) satisfying two conditions:

1. For every tree T, and accepting runs ρ_1 and ρ_2, we have $S_{\rho_1}(T) = S_{\rho_2}(T)$; and
2. The automaton (Q, F, δ) accepts every tree.

For such QAs, we can unambiguously define the set of selected nodes as $\mathcal{QA}(T) = S_\rho(T)$, where ρ is an arbitrarily chosen accepting run.

While the conditions are fairly strong, they do not restrict the power of QAs:

Fact 1 (see [15, 32, 33]) *For every query automaton $\mathcal{QA}$, there exists an equivalent single-run query automaton, that is, a single-run query automaton $\mathcal{QA}'$ such that $\mathcal{QA}^{\exists}(T) = \mathcal{QA}'(T)$ for every tree T.*

Remarks: the construction in [15] needs a slight modification to produce such QA; also it needs to be extended to unranked trees which is straightforward. This was also noticed in [33]. One can also get this result by slightly adapting the construction of [32].

We now make a few remarks about closure properties and decision problems for single-run QAs. It is known [29] that nonemptiness problem for existential-semantics QAs is solvable in polynomial time; hence the same is true for single-run QAs. Single-run QAs are easily closed under intersection: the usual product construction works. Moreover, if one takes a product $\mathcal{A} \times \mathcal{QA}$ of a tree automaton and a single-run QA (where selecting states are pairs containing a selecting state of $\mathcal{QA}$), the result is a QA satisfying 1) above, and the nonemptiness problem for it is solvable in polynomial time too.

We define the *complement* of a single-run QA as $\overline{\mathcal{QA}} = (Q, F, Q - Q_s, \delta)$, where $\mathcal{QA} = (Q, F, Q_s, \delta)$. It follows immediately from the definition that for every tree T with domain D, we have $\overline{\mathcal{QA}}(T) = D - \mathcal{QA}(T)$, if $\mathcal{QA}$ is single-run. This implies that the *containment problem* $\mathcal{QA}_1 \subseteq \mathcal{QA}_2$ (i.e., checking whether $\mathcal{QA}_1(T) \subseteq \mathcal{QA}_2(T)$ for all T) for single-run QAs is solvable in polynomial time, since it is equivalent to checking emptiness of $\mathcal{QA}_1 \times \overline{\mathcal{QA}_2}$.

4 Logics on trees: $\mathrm{TL}^{\mathrm{tree}}$ and XPath

$\mathbf{TL}^{\mathbf{tree}}$ An unranked tree $T = (D, \lambda)$ can be viewed as a structure $\langle D, \prec^*_{\mathrm{ch}}, \prec^*_{\mathrm{ns}}, (P_a)_{a \in \Sigma} \rangle$, where P_a's are labeling predicates: $P_a = \{s \in D \mid \lambda(s) = a\}$. Thus, when we talk about *first-order logic (*FO*)*, or *monadic second-order logic (*MSO*)*, we interpret them on these representations of unranked trees. Recall that MSO extends FO with quantification over sets.

We shall use a *tree temporal logic* [26, 34], denoted here by $\mathrm{TL}^{\mathrm{tree}}$ [22]. It can be viewed as a natural extension of LTL with the past operators to unranked trees [20, 38], with *next*, *previous*, *until*, and *since* operators for both child and next-sibling relations. The syntax of $\mathrm{TL}^{\mathrm{tree}}$ is defined by:

$$\varphi, \varphi' \ := \ \top \mid \bot \mid a \mid \varphi \vee \varphi' \mid \neg\varphi \mid \mathbf{X}_*\varphi \mid \mathbf{X}^-_*\varphi \mid \varphi \mathbf{U}_* \varphi' \mid \varphi \mathbf{S}_* \varphi',$$

where $\top$ and $\bot$ are true and false, a ranges over Σ, and $*$ is either 'ch' (child) or 'ns' (next sibling). The semantics is defined with respect to a tree $T = (D, \lambda)$ and a node $s \in D$:

- $(T, s) \models \top$; $\quad (T, s) \not\models \bot$;
- $(T, s) \models a$ iff $\lambda(s) = a$;
- $(T, s) \models \varphi \vee \varphi'$ iff $(T, s) \models \varphi$ or $(T, s) \models \varphi'$;
- $(T, s) \models \neg\varphi$ iff $(T, s) \not\models \varphi$;
- $(T, s) \models \mathbf{X}_{\text{ch}}\varphi$ if there exists a node $s' \in D$ such that $s \prec_{\text{ch}} s'$ and $(T, s') \models \varphi$;
- $(T, s) \models \mathbf{X}^{-}_{\text{ch}}\varphi$ if there exists a node $s' \in D$ such that $s' \prec_{\text{ch}} s$ and $(T, s') \models \varphi$;
- $(T, s) \models \varphi \mathbf{U}_{\text{ch}} \varphi'$ if there is a node s' such that $s \prec^*_{\text{ch}} s'$, $(T, s') \models \varphi'$, and for all $s'' \neq s'$ satisfying $s \prec^*_{\text{ch}} s'' \prec^*_{\text{ch}} s'$ we have $(T, s'') \models \varphi$.
- $(T, s) \models \varphi \mathbf{S}_{\text{ch}} \varphi'$ if there is a node s' such that $s' \prec^*_{\text{ch}} s$, $(T, s') \models \varphi'$, and for all $s'' \neq s'$ satisfying $s' \prec^*_{\text{ch}} s'' \prec^*_{\text{ch}} s$ we have $(T, s'') \models \varphi$.

The semantics of $\mathbf{X}_{\text{ns}}, \mathbf{X}^{-}_{\text{ns}}, \mathbf{U}_{\text{ns}}$, and $\mathbf{S}_{\text{ns}}$ is analogous by replacing the child relation with the next-sibling relation.

A TL^{tree} formula φ defines a unary query $T \mapsto \{s \mid (T, s) \models \varphi\}$. It is known that TL^{tree} is expressively complete for FO: the class of such unary queries is precisely the class of queries defined by FO formulae with one free variable [26, 34].

XPath We present a first-order complete extension of XPath, called *conditional XPath*, or CXPath [26]. We introduce very minor modifications to the syntax (e.g., we use an existential quantifier $\mathbf{E}$ instead of the usual XPath node test brackets [ ]) to make the syntax resemble that of temporal logics. CXPath has *node formulae* α and *path formulae* β given by:

$$\alpha, \alpha' := a \mid \neg\alpha \mid \alpha \vee \alpha' \mid \mathbf{E}\beta$$
$$\beta, \beta' := ?\alpha \mid \texttt{step} \mid \texttt{step}^* \mid (\texttt{step}/?\alpha)^* \mid \beta/\beta' \mid \beta \vee \beta'$$

where a ranges over Σ and $\texttt{step}$ is one of the following: $\prec_{\text{ch}}$, $\prec^{-}_{\text{ch}}$, $\prec_{\text{ns}}$, or $\prec^{-}_{\text{ns}}$. The language without the $(\texttt{step}/?\alpha)^*$ is known as "core XPath".

Intuitively $\mathbf{E}\beta$ states the existence of a path starting in a given node and satisfying β, the path formula $?\alpha$ tests if the node formula α is true in the initial node of a path, and / is the composition of paths.

Given a tree $T = (D, \lambda)$, the semantics of a node formula is a set of nodes $[\![\alpha]\!]_T \subseteq D$, and the semantics of a path formula is a binary relation $[\![\beta]\!]_T \subseteq D \times D$ given by the following rules. We use R^* to denote the reflexive-transitive closure of relation R, and $\pi_1(R)$ to denote its first projection.

$$[\![a]\!]_T = \{s \in D \mid \lambda(s) = a\} \qquad [\![?\alpha]\!]_T = \{(s, s) \mid s \in [\![\alpha]\!]_T\}$$
$$[\![\neg\alpha]\!]_T = D - [\![\alpha]\!]_T \qquad [\![\texttt{step}]\!]_T = \{(s, s') \mid s, s' \in D \text{ and } (s, s') \in \texttt{step}\}$$
$$[\![\alpha \vee \alpha']\!]_T = [\![\alpha]\!]_T \cup [\![\alpha']\!]_T \qquad [\![\beta \vee \beta']\!]_T = [\![\beta]\!]_T \cup [\![\beta']\!]_T$$
$$[\![\mathbf{E}\beta]\!]_T = \pi_1([\![\beta]\!]_T) \qquad [\![\texttt{step}^*]\!]_T = [\![\texttt{step}]\!]^*_T$$
$$[\![\beta/\beta']\!]_T = [\![\beta]\!]_T \circ [\![\beta']\!]_T$$
$$[\![(\texttt{step}/?\alpha)^*]\!]_T = [\![(\texttt{step}/?\alpha)]\!]^*_T$$

CXPath defines two kinds of unary queries: those given by node formulae, and those given by path formulae β, selecting $[\![\beta]\!]_T^{root} = \{s \in D \mid (\varepsilon, s) \in [\![\beta]\!]_T\}$. Both classes capture precisely unary FO queries on trees [26].

XPath and TL$^{\mathbf{tree}}$ XPath expressions can be translated into TL$^{\text{tree}}$. For example, consider an expression in the "traditional" XPath syntax: $e = /a//b[//c]$. It says: start at the root, find children labeled a, their descendants labeled b, and select those which have a c-descendant. It can be viewed as both a path formula and a node formula of XPath. An equivalent path formula is

$$\beta \;= \prec_{\text{ch}} /?a/\, \prec_{\text{ch}}^* /?(b \wedge \mathbf{E}(\prec_{\text{ch}}^* /?c)).$$

The set $[\![\beta]\!]_T^{root} = \{s \mid (\varepsilon, s) \in [\![\beta]\!]_T\}$ is precisely the set of nodes selected by e in T. Alternatively we can view it as a node formula

$$\alpha = b \wedge \mathbf{E}(\prec_{\text{ch}}^* /?c) \wedge \mathbf{E}\big((\prec_{\text{ch}}^-)^* /?(a \wedge \mathbf{E}(\prec_{\text{ch}}^- /root))\big).$$

Here *root* is an abbreviation for a formula that tests for the root node. Then $[\![\alpha]\!]_T$ generates the set of nodes selected by e. It is known [27] that for every path formula β, one can construct in linear time a node formula α so that $[\![\beta]\!]_T^{root} = [\![\alpha]\!]_T$. Thus, from now on we deal with node XPath formulae.

The above formulae can be translated into an equivalent TL$^{\text{tree}}$ expression

$$b \wedge \mathbf{F}_{\text{ch}} c \wedge \mathbf{F}_{\text{ch}}^- \big(a \wedge \mathbf{X}_{\text{ch}}^- root\big)$$

Here $\mathbf{F}_{\text{ch}}\varphi$ is $\top \mathbf{U}_{\text{ch}}\varphi$, and $\mathbf{F}_{\text{ch}}^-\varphi$ is $\top \mathbf{S}_{\text{ch}}\varphi$; we also use *root* as a shorthand for $\neg \mathbf{X}_{\text{ch}}^- \top$. This formula selects b-labeled nodes with c-labeled descendants, and an a-ancestor which is a child of the root – this is of course equivalent to the original expression.

Since both TL$^{\text{tree}}$ and CXPath are first-order expressively-complete [26], each core or conditional XPath expression is equivalent to a formula of TL$^{\text{tree}}$; however, no direct translation has previously been produced. We now give such a direct translation that, together with the translation from TL$^{\text{tree}}$ to QAs, will guarantee single-exponential bounds on QAs equivalent to XPath formulae.

Lemma 1. *There is a translation of node formulae α of core or conditional XPath into formulae α' of* TL$^{\text{tree}}$ *such that the number of subformulae of α' is at most linear in the size of α. Moreover, if α does not use any disjunctions of path formulae, then the size of α' is at most linear in the size of α.*

In particular, even if α' is exponential in the size of α, the size of its Fischer-Ladner closure is at most linear in the size of the original formula α.

We now sketch the proof. Given two TL$^{\text{tree}}$ formulae φ and φ' and a CXPath path formula β, we write $\varphi' \equiv \mathbf{X}_\beta \varphi$ if for each tree T and each node s, one has that $(T, s) \models \varphi'$ iff there is a node s', with $(s, s') \in [\![\beta]\!]_T$, such that $(T, s') \models \varphi$. Now each CXPath node formula α is translated into a TL$^{\text{tree}}$ formula φ_α such that $(T, s) \models \varphi_\alpha$ iff $s \in [\![\alpha]\!]_T$. Each path formula β is translated into a mapping

x_β from $\mathrm{TL}^{\mathrm{tree}}$ formulae to $\mathrm{TL}^{\mathrm{tree}}$ formulae such that $x_\beta(\varphi) \equiv \mathbf{X}_\beta\varphi$. The rules are:

α	φ_α
a	a
$\neg\alpha'$	$\neg\varphi_{\alpha'}$
$\alpha' \vee \alpha''$	$\varphi_{\alpha'} \vee \varphi_{\alpha''}$
$\mathbf{E}\beta$	$x_\beta(\top)$

β	$x_\beta(\varphi)$
$?\alpha$	$\varphi_\alpha \wedge \varphi$
$\prec_{\mathrm{ch}}$	$\mathbf{X}_{\mathrm{ch}}\varphi$
$\prec^*_{\mathrm{ch}}$	$\top\mathbf{U}_{\mathrm{ch}}\varphi$
$(\prec_{\mathrm{ch}}/?\alpha)^*$	$(\mathbf{X}_{\mathrm{ch}}\varphi_\alpha)\mathbf{U}_{\mathrm{ch}}\varphi$
β'/β''	$x_{\beta'} \circ x_{\beta''}(\varphi)$
$\beta \vee \beta'$	$x_{\beta'}(\varphi) \vee x_{\beta''}(\varphi)$

□

5 Tree logic into query automata: a translation

Our goal is to translate $\mathrm{TL}^{\mathrm{tree}}$ into single-run QAs. We do a direct translation into unranked QAs, as opposed to coding of unranked trees into binary (which is a common technique). Such coding is problematic for two reasons. First, simple navigation over unranked trees may look unnatural when coded into binary, resulting in more complex formulae (child, for example, becomes 'left successor followed by zero or more right successors'). Second, coding into binary trees makes reasoning about views much harder. The property of being upward-closed, which is essential for decidability of certain answers, is not even preserved by the translation. Thus, we do a direct translation into unranked QAs, and then apply it to XML specifications.

Since values of transitions $\delta(q, a)$ in unranked QAs are not sets of states but rather NFAs representing regular languages over states, we measure the size of $\mathcal{QA} = (Q, F, Q_s, \delta)$ not as the number $|Q|$ of states, but rather as

$$\|\mathcal{QA}\| = |Q| + \sum_{q\in Q, a\in\Sigma} \|\delta(q, a)\|,$$

where $\|\delta(q, a)\|$ is the number of states of the NFA. We then show:

Theorem 1. *Every* $\mathrm{TL}^{\mathrm{tree}}$ *formula* φ *of size* n *can be translated, in exponential time, into an equivalent single-run query automaton* $\mathcal{QA}_\varphi$ *of size* $2^{O(n)}$*, i.e. a query automaton such that* $\mathcal{QA}_\varphi(T) = \{s \mid (T, s) \models \varphi\}$ *for every tree* T*.*

We now sketch the construction. First, as is common with translations into nondeterministic automata [39], we need to work with a version of $\mathrm{TL}^{\mathrm{tree}}$ in which all negations are pushed to propositions. To deal with until and since operators, we shall introduce four operators $\mathbf{R}_*$ and $\mathbf{I}_*$ for $*$ being 'ch' or 'ns' so that $\neg(\alpha\mathbf{U}_*\beta) \leftrightarrow \neg\alpha\mathbf{R}_*\neg\beta$ and $\neg(\alpha\mathbf{S}_*\beta) \leftrightarrow \neg\alpha\mathbf{I}_*\neg\beta$; this part is completely standard. However, trees do not have a linear structure and we cannot just push negation inside the $\mathbf{X}$ operators: for example, $\neg\mathbf{X}_{\mathrm{ch}}\varphi$ is not $\mathbf{X}_{\mathrm{ch}}\neg\varphi$. Since our semantics of the next operators is existential (there is a successor node in which the formula is true), we need to add their universal analogs. For example, $\mathbf{X}^\forall_{\mathrm{ch}}\varphi$

is true in s if for every successor s' of s in the domain of the tree, φ is true in s'. Then of course we have $\neg \mathbf{X}_{\mathrm{ch}} \varphi \leftrightarrow \mathbf{X}^{\forall}_{\mathrm{ch}} \neg \varphi$. We add four such operators ($\mathbf{X}^{\forall}_{\mathrm{ch}}, \mathbf{X}^{\forall}_{\mathrm{ns}}, \mathbf{X}^{-\forall}_{\mathrm{ch}}, \mathbf{X}^{-\forall}_{\mathrm{ns}}$). Other axes have a linear structure, so one could alternatively add tests for the root, first, and last child of a node to deal with them. For example, $\neg \mathbf{X}^{-}_{\mathrm{ch}} \varphi \leftrightarrow \mathbf{X}^{-}_{\mathrm{ch}} \neg \varphi \vee \alpha_{root}$, where α_{root} is a test for the root. But for symmetry we prefer to deal with the four universal versions of the next/previous operators, since it is unavoidable for $\mathbf{X}_{\mathrm{ch}}$.

With these additions, we can push negations to propositions, so we assume negations only occur in subformulae $\neg a$ for $a \in \Sigma$. The states of $\mathcal{QA}_\varphi$ will be maximally consistent subsets of the Fischer-Ladner closure of φ (in particular, for each state q and a subformula ψ, exactly one of ψ and $\neg\psi$ is in q).

The transitions have to ensure that all "horizontal" temporal connectives behave properly, and that "vertical" transitions are consistent. The alphabet of each automaton $\delta(q, a)$ is the set of states of $\mathcal{QA}_\varphi$; that is, letters of $\delta(q, a)$ are sets of formulae. Each $\delta(q, a)$ is a product of three automata. The first guarantees that eventualities $\alpha \mathbf{U}_{\mathrm{ns}} \beta$ and $\alpha \mathbf{S}_{\mathrm{ns}} \beta$ are fulfilled in the oldest and youngest siblings. For that, we impose conditions on the initial states $\delta(q, a)$'s that they need to read a letter (which is a state of $\mathcal{QA}_\varphi$) that may not contain $\alpha \mathbf{S}_{\mathrm{ns}} \beta$ without containing β, and on their final state guaranteeing that in the last letter we do not have a subformula $\alpha \mathbf{U}_{\mathrm{ns}} \beta$ without having β.

The second automaton enforces horizontal transitions, and it behaves very similarly to the standard LTL-to-Büchi construction; it only deals with next-sibling connectives. For example, if $\mathbf{X}_{\mathrm{ns}} \alpha$ is the current state of $\mathcal{QA}$ for a node $s \cdot i$, then the state for $s \cdot (i + 1)$ contains α, and that if $\alpha \mathbf{U}_{\mathrm{ns}} \beta$ is in the state for $s \cdot i$ but β is not, then $\alpha \mathbf{U}_{\mathrm{ns}} \beta$ is propagated into the state for $s \cdot (i + 1)$.

The third automaton enforces vertical transitions. We give a few sample rules. If q contains the negation of $\alpha \mathbf{S}_{\mathrm{ch}} \beta$, then the automaton rejects after seeing a state which contains $\alpha \mathbf{S}_{\mathrm{ch}} \beta$ but does not contain β (since in this case $\alpha \mathbf{S}_{\mathrm{ch}} \beta$ must propagate to the parent). If q contains $\alpha \mathbf{U}_{\mathrm{ch}} \beta$ and does not contain β, then the automaton only accepts if one of its input letters contains $\alpha \mathbf{U}_{\mathrm{ch}} \beta$. And if q contains $\mathbf{X}_{\mathrm{ch}} \alpha$, then it only accepts if one of its input letters contains α. In addition, we have to enforce eventualities $\alpha \mathbf{U}_{\mathrm{ch}} \beta$ by disallowing these automata to accept ε if q contains $\alpha \mathbf{U}_{\mathrm{ch}} \beta$ and does not contain β.

The final states of $\mathcal{QA}_\varphi$ at the root must enforce correctness of $\alpha \mathbf{S}_{\mathrm{ch}} \beta$ formulae: with each such formula, states from F must contain β as well. This completes the construction. When all automata $\delta(q, a)$ are properly coded, the $2^{O(n)}$ bound follows. We then show a standard lemma that in an accepting run, a node is assigned a state that contains a subformula α iff α is true in that node. This guarantees that for every tree, there is an accepting run. Since each state has either α or $\neg\alpha$ in it, it follows that the resulting QA is single-run.

6 An application: reasoning about document navigation

As mentioned in Section 2, typical XML static analysis tasks include consistency of schema and navigational properties (e.g., is a given XPath expression

consistent with a given DTD?), or query optimization (e.g., is a given XPath expression e contained in a another expression e' for all trees that conform to a DTD d?). We now show two applications of our results for such analyses of XML specifications.

Satisfiability algorithms for sets of XPath expressions The exponential-time complexity for satisfiability of XPath expressions in the presence of a schema is already known [25, 6]. We now show how we can verify satisfiability of multiple sets of XPath expressions, in a uniform way, using translation into query automata.

Given an arbitrary set $E = \{e_1, \ldots, e_n\}$ of XPath (core or conditional) expressions and a subset $E' \subseteq E$, let $\mathcal{Q}(E')$ be a unary query defining the intersection of queries given by all the $e \in E'$. That is, $\mathcal{Q}(E')$ selects nodes that satisfy every expression $e \in E'$. We can capture *all* (exponentially many) such queries $\mathcal{Q}(E')$s by a single automaton, that is instantiated into different QAs by different selecting states.

Corollary 1. *One can construct, in time* $2^{O(\|E\|)}$ *(that is,* $2^{O(\|e_1\|+\ldots+\|e_n\|)}$*), an unranked tree automaton* $\mathcal{A}(E) = (Q, F, \delta)$ *and a relation* $\sigma \subseteq E \times Q$ *such that, for every* $E' \subseteq E$*,*

$$\mathcal{QA}_{E'} \;=\; (Q,\; F,\; \bigcap\{\sigma(e) \mid e \in E'\}, \delta)$$

is a single-run QA defining the unary query $\mathcal{Q}(E')$*.*

The construction simply takes the product of all the $\mathcal{QA}_{e'_i}$s, produced by Theorem 1, where e'_i is a $\mathrm{TL}^{\mathrm{tree}}$ translation of e_i, produced by Lemma 1. The relation σ relates tuples of states that include selecting states of $\mathcal{QA}_{e'_i}$ with $e_i \in E$. Then checking nonemptiness of $\mathcal{QA}_{E'}$, we see if all $e \in E'$ are simultaneously satisfiable.

The containment problem for XPath expressions is a special case of the problem we consider. To check whether $d \models e_1 \subseteq e_2$, we construct $\mathcal{QA}_{\{e_1, \neg e_2\}}$ as in Corollary 1, and take the product of it with the automaton for d. This results in a QA of size $\|d\| \cdot 2^{O(\|e_1\|+\|e_2\|)}$ that finds counterexamples to containment under d. This is precisely the construction that was promised in the introduction.

Verifying complex containment statements under DTDs We can now extend the previous example and check not a single containment, as is usually done [35], but arbitrary Boolean combinations of XPath containment statements, without additional complexity. Assume that we are given a DTD d (or any other schema specification presented by an automaton), a set $\{e_1, \ldots, e_n\}$ of XPath expressions, and a Boolean combination $\mathcal{C}$ of inclusions $e_i \subseteq e_j$. We now want to check whether $d \models \mathcal{C}$, that is, whether $\mathcal{C}$ is true in every tree T that conforms to d. We shall refer to size of $\mathcal{C}$ as $\|\mathcal{C}\|$; the definition is extended in the natural way from the definition of $\|e\|$.

Theorem 2. *In the above setting, one can construct an unranked tree automaton of size* $\|d\| \cdot 2^{O(\|\mathcal{C}\|)}$ *whose language is empty iff* $d \models \mathcal{C}$*.*

This is achieved by replacing $e_i \subseteq e_j$ in $\mathcal{C}$ with the formula $\neg\mathbf{F}_{\mathrm{ch}}(e'_i \wedge \neg e'_j)$ and $e_i \not\subseteq e_j$ in $\mathcal{C}$ with the formula $\mathbf{F}_{\mathrm{ch}}(e'_i \wedge \neg e'_j)$, where e'_i, e'_j are $\mathrm{TL}^{\mathrm{tree}}$ translations of e_i and e_j produced by Lemma 1. Thus we can view $\mathcal{C}$ as a $\mathrm{TL}^{\mathrm{tree}}$ formula $\alpha_{\mathcal{C}}$. Now construct a QA for $\neg\alpha_{\mathcal{C}}$, by Theorem 1, and turn it into an automaton that checks whether the root gets selected. Now we take the product of this automaton with the automaton for d. The result accepts counterexamples to $\mathcal{C}$ under d, and the result follows. The construction of the automaton is polynomial-time in $\|d\|$ and single-exponential time in $\|\mathcal{C}\|$.

7 An application: reasoning about views

Recall the problem outlined in the introduction. We have a view definition given by a query automaton $\mathcal{QA}_{\mathcal{V}}$. For each source tree T, it selects a set of nodes $V = \mathcal{QA}_{\mathcal{V}}(T)$ which can also be viewed as a tree (we can assume, for example, that $\mathcal{QA}_{\mathcal{V}}$ always selects the root). Source trees are required to satisfy a schema constraint (e.g., a DTD). Since all schema formalisms for XML are various restrictions or reformulations of tree automata, we assume that the schema is given by an automaton $\mathcal{A}$.

If we only have access to V, we would like to be sure that secret information about an unknown source T is not revealed. This information, which we assume to be coded by a Boolean query $\mathfrak{Q}$, would be revealed by V if the answer to $\mathfrak{Q}$ were true in all source trees T that conform to the schema and generate V – that is, if $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V)$ were true. Thus, we would like to construct a new automaton $\mathcal{A}^*$ that accepts V iff $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V)$ is false, giving us some security assurances about the view.

In general, such an automaton construction is impossible: if $\mathcal{QA}_{\mathcal{V}}$ generates the yield of a tree, views essentially code context-free languages. Combining multiple CFLs with the help of DTDs, we get an undecidability result:

Proposition 1. *The problem of checking, for source and view schemas $\mathcal{A}_s$ and $\mathcal{A}_v$, a view definition $\mathcal{QA}_{\mathcal{V}}$, and a Boolean first-order query $\mathfrak{Q}$, whether there exists a view V that conforms to $\mathcal{A}_v$ and satisfies $\underline{certain}^{\mathcal{A}_s}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V) = \mathrm{true}$, is undecidable.*

Schemas and queries required for this result are very simple, so to ensure the existence of the automaton $\mathcal{A}^*$, we need to put restrictions on the class of views. We assume that they are *upward-closed* as in [7]: if a node is selected, then so is the entire path to it from the root.

Note that the upward-closure $\mathcal{QA}_{\uparrow}$ of a query automaton $\mathcal{QA}$ can be obtained in linear time by adding a bit to the state indicating whether a selecting state has been seen and propagating it up. Thus, we shall assume without loss of generality that QAs defining views are *upward-closed*: if $s \in \mathcal{QA}(T)$ and s' is an ancestor of s, then $s' \in \mathcal{QA}(T)$.

The key observation that we need is that for an upward-closed QA, satisfying the single-run condition, its image is regular. Furthermore, it can be accepted by a small tree automaton:

Lemma 2. *Let $\mathcal{QA}$ be an upward-closed query automaton that satisfies condition 1) of the definition of single-run QAs. Then one can construct, in cubic time, an unranked tree automaton $\mathcal{A}^*$ that accepts trees V for which there exists a tree T satisfying $V = \mathcal{QA}(T)$. Moreover, the number of states of $\mathcal{A}^*$ is at most the number of states of $\mathcal{QA}$.*

Proof sketch. The automaton $\mathcal{A}^*$ has to guess a tree T and its run so that the selecting states would be assigned precisely to the elements of V. So one first needs to analyze non-selecting runs: that is, runs that can be extended to an accepting run but never hit a selecting state. Trees admitting such runs may be inserted under leaves of V, and in between two consecutive siblings of a node in V. We then need to modify the horizontal transition to allow for guesses of words consisting of final states of non-selecting runs in between two states. □

To apply Lemma 2 to the problem of finding certain answers $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V)$, we now take the product of $\mathcal{QA}_\mathcal{V}$ with $\mathcal{A}$ and the automaton for $\neg\mathfrak{Q}$ (the selecting states in the product will be determined by $\mathcal{QA}_\mathcal{V}$), and obtain:

Theorem 3. *Let $\mathcal{QA}_\mathcal{V}$ be upward-closed and single-run, $\mathcal{A}$ an unranked tree automaton defining a schema, and $\mathcal{A}_{\neg\mathfrak{Q}}$ an automaton accepting trees for which $\mathfrak{Q}$ is false. Then one can construct, in polynomial time, an unranked tree automaton $\mathcal{A}^*$ such that*

1. $\|\mathcal{A}^*\| = O(\|\mathcal{QA}_\mathcal{V}\| \cdot \|\mathcal{A}\| \cdot \|\mathcal{A}_{\neg\mathfrak{Q}}\|)$, *and*
2. $\mathcal{A}^*$ accepts $V \quad \Leftrightarrow \quad \underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V) = \text{false}$.

Combining Theorem 3 with previous translations into single-run QAs and properties of the latter, we obtain algorithms for verifying properties of views given by XPath expressions. Revisiting our motivating example from Section 2, we make the following assumptions:

- The view definition is given by an XPath (conditional or core) expression e_V; the view V of a source tree T has all the nodes selected by e_V and their ancestors;
- The schema definition is given by a DTD d;
- The query $\mathfrak{Q}$ is an arbitrary Boolean combination of containment statements $e \subseteq e'$, where e, e' come from a set E of XPath expressions.

Then, for a given V, we want to check if $\underline{certain}^{d}_{e_V}(\mathfrak{Q}; V)$ is false: that is, the secret encoded by $\mathfrak{Q}$ cannot be revealed by V, since not all source trees T that conform to d and generate V satisfy $\mathfrak{Q}$. We then have the following:

Corollary 2. *In the above setting, one can construct in time polynomial in $\|d\|$ and exponential in $\|E\| + \|e_V\|$ an unranked tree automaton $\mathcal{A}^*$ of size $\|d\| \cdot 2^{O(\|e_V\|+\|E\|)}$ that accepts a view V iff $\underline{certain}^{d}_{e_V}(\mathfrak{Q}; V)$ is false.*

Note that again the exponent contains the size of typically small XPath expressions, and not the potentially large schema definition d.

8 Conclusion

There are several extensions we would like to consider. One concerns relative specifications often used in the XML context – these apply to subtrees. Results of [21, 2] on model-checking of *now* and *within* operators on words and nested words indicate that an exponential blowup is unavoidable, but there could well be relevant practical cases that do not exhibit it. We would like to see how LTL-to-Büchi optimization techniques (e.g., in [12, 17]) could be adapted in our setting, to produce automata of smaller size. We also would like to see if automata can be used for reasoning about views without imposing upward-closeness of [7], which does not account for some of the cases of secure XML views [13]. One could look beyond first-order at logics having the power of MSO or ambient logics with known translations into automata, and investigate their translations into QAs [9, 18, 15]. Another possible direction has to do with a SAX representation of XML which corresponds to its linear structure (in the paper we dealt with the tree structure, i.e., the DOM representation). The connection between the linear structure of XML and nested words already found some applications [19, 23].

Acknowledgment We thank Pablo Barceló and Floris Geerts for their comments. This work was done while the second author was at the University of Edinburgh. The authors were supported by EPSRC grant E005039, the first author also by the European Commission Marie Curie Excellence grant MEXC-CT-2005-024502.

References

1. S. Abiteboul, B. Cautis, T. Milo. Reasoning about XML update constraints. In *PODS'07*, pages 195–204.
2. R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, L. Libkin. First-order and temporal logics for nested words. In *LICS'07*, pages 151–160.
3. R. Alur, K. Etessami and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS'04*, pages 467–481.
4. M. Arenas, W. Fan, L. Libkin. Consistency of XML specifications. In *Inconsistency Tolerance*, Springer, 2005, pages 15–41.
5. P. Barceló, L. Libkin. Temporal logics over unranked trees. In *LICS'05*, pages 31–40.
6. M. Benedikt, W. Fan, F. Geerts. XPath satisfiability in the presence of DTDs. In *PODS'05*, pages 25–36.
7. M. Benedikt and I. Fundulaki. XML subtree queries: specification and composition. In *DBPL'05*, pages 138–153.
8. M. Bojanczyk, C. David, A. Muscholl, Th. Schwentick, L. Segoufin. Two-variable logic on data trees and XML reasoning. In *PODS'06*, pages 10–19.
9. I. Boneva, J.-M. Talbot, S. Tison. Expressiveness of a spatial logic for trees. In *LICS 2005*, pages 280–289.
10. D. Calvanese, G. De Giacomo, M. Lenzerini, M. Y. Vardi. Regular XPath: constraints, query containment and view-based answering for XML documents. In *Logic in Databases*, 2008.
11. E. Clarke, O. Grumberg, D. Peled. *Model Checking*, MIT Press, 1999.

12. M. Daniele, F. Giunchiglia, M.Y. Vardi. Improved automata generation for linear temporal logic. In *CAV'99*, pages 249–260.
13. W. Fan, F. Geerts, X. Jia, A. Kementsietsidis. Rewriting regular XPath queries on XML views. In *ICDE'07*, pages 666–675.
14. W. Fan, C.Y. Chan, M. Garofalakis. Secure XML querying with security views. In *SIGMOD'04*, pages 587–598.
15. M. Frick, M. Grohe, C. Koch. Query evaluation on compressed trees. In *LICS'03*, pages 188-197.
16. P. Genevés and N. Layaida. A system for the static analysis of XPath. *ACM TOIS* 24 (2006), 475–502.
17. R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV 1995*, pages 3–18.
18. G. Gottlob, C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM* 51 (2004), 74–113.
19. V. Kumar, P. Madhusudan, M. Viswanathan. Visibly pushdown automata for streaming XML. In *WWW 2007*, pages 1053–1062.
20. O. Kupferman, A. Pnueli. Once and for all. In *LICS'95*, pages 25–35.
21. F. Laroussinie, N. Markey, Ph. Schnoebelen. Temporal logic with forgettable past. In *LICS'02*, pages 383–392.
22. L. Libkin. Logics for unranked trees: an overview. In *ICALP'05*, pages 35-50.
23. P. Madhusudan, M. Viswanathan. Query automata for nested words. Manuscript, 2008.
24. S. Maneth, T. Perst, H. Seidl. Exact XML type checking in polynomial time. In *ICDT 2007*, pages 254–268.
25. M. Marx. XPath with conditional axis relations. In *EDBT 2004*, pages 477–494.
26. M. Marx. Conditional XPath. *ACM TODS* 30 (2005), 929–959.
27. M. Marx, M. de Rijke. Semantic characterizations of navigational XPath. *SIGMOD Record* 34 (2005), 41–46.
28. F. Neven. *Design and Analysis of Query Languages for Structured Documents*. PhD Thesis, U. Limburg, 1999.
29. F. Neven. Automata, logic, and XML. In *CSL 2002*, pages 2–26.
30. F. Neven, Th. Schwentick. Query automata over finite trees. *TCS*, 275 (2002), 633–674.
31. F. Neven, Th. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *LMCS*, 2(3): (2006).
32. F. Neven, J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. *J. ACM* 49(1): 56–100 (2002).
33. J. Niehren, L. Planque, J.-M. Talbot, S. Tison. N-ary queries by tree automata. In *DBPL 2005*, pages 217–231.
34. B.-H. Schlingloff. Expressive completeness of temporal logic of trees. *Journal of Applied Non-Classical Logics* 2 (1992), 157–180.
35. Th. Schwentick. XPath query containment. *SIGMOD Record* 33 (2004), 101–109.
36. Th. Schwentick. Automata for XML – a survey. *JCSS* 73 (2007), 289–315.
37. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. Banff Higher Order Workshop, 1996.
38. M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP'98*, pages 628–641.
39. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf.& Comput.* 115 (1994), 1–37.
40. M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *JCSS* 33 (1986), 183–221.