Combining Temporal Logics for Querying XML Documents

Marcelo Arenas[1], Pablo Barceló[2], and Leonid Libkin[3]

[1] Pontificia Universidad Católica de Chile
[2] Universidad de Chile
[3] University of Edinburgh

Abstract. Close relationships between XML navigation and temporal logics have been discovered recently, in particular between logics LTL and CTL* and XPath navigation, and between the μ-calculus and navigation based on regular expressions. This opened up the possibility of bringing model-checking techniques into the field of XML, as documents are naturally represented as labeled transition systems. Most known results of this kind, however, are limited to Boolean or unary queries, which are not always sufficient for complex querying tasks.

Here we present a technique for combining temporal logics to capture n-ary XML queries expressible in two yardstick languages: FO and MSO. We show that by adding simple terms to the language, and combining a temporal logic for words together with a temporal logic for unary tree queries, one obtains logics that select arbitrary tuples of elements, and can thus be used as building blocks in complex query languages. We present general results on the expressiveness of such temporal logics, study their model-checking properties, and relate them to some common XML querying tasks.

1 Introduction

It has been observed many times that the basic settings of the fields of database querying and model checking are very similar: in both cases one needs to evaluate a logical formula on a finite relational structure. Both fields have invested heavily in developing logical formalisms and efficient algorithms for query evaluation and model checking, but despite this, there are very few direct connections between them, although there is certainly interest in bringing them closer together (see, e.g., an invited talk at the last ICDT [36]).

Our goal is to explore one possible connection between database querying and temporal-logic model-checking: we concentrate on the recently discovered connections between XML querying/navigation, and temporal and modal logics [1,3,6,25,15,27]. Since XML documents are modeled as labeled unranked trees with a sibling ordering [19,28], they can naturally be viewed as labeled transition systems. Furthermore, many common XML tasks involve navigation via paths in a document, reminiscent of temporal properties of paths in transition systems.

In terms of expressiveness, the yardstick logics for XML querying are FO (first-order) and MSO (monadic second-order). But from the point of view of

T. Schwentick and D. Suciu (Eds.): ICDT 2007, LNCS 4353, pp. 359–373, 2007.

efficiency of query evaluation, they are not the best, as they cannot guarantee fast (linear-time) query-evaluation – which is often the goal for query evaluation on trees [20] – without a very high (nonelementary) price in terms of the size of the query [14]. However, many temporal logics overcome this problem [20,25,3], which makes them suitable for XML querying.

The connection between XML navigation and temporal logics was best demonstrated in the work of Marx [25] and his followers [7,1,2,15]. In particular, [25] gave an expressive completeness result for XPath: adding a temporal *until* operator (found in logics such as LTL, CTL) to the core of XPath gives it precisely the power of FO, one of the yardstick database query languages. FO sentences over both binary and unranked trees are also known [18,3] to have the power of a commonly used temporal logic CTL*, and MSO has the power of the modal μ-calculus over both binary [30] and unranked trees [3].

The main limitation of these results is that they only apply to Boolean (i.e., yes/no) queries, or unary queries, that select a set of nodes from a document (and the result of [25] also extends to queries with two free variables). While for problems such as validation, or for some information extraction tasks [16] this is sufficient, there are many cases where more expressiveness is needed than Boolean or unary queries provide. For example, the core of XQuery consists of expressions that essentially select arbitrary tuples of nodes, based on properties of paths leading to them, and then output them rearranged as a different tree. But while it is known that the usual MSO/automata connection extends to the case of n-ary queries [31], logical formalisms for n-ary queries and their model-checking properties have not been adequately explored.

In this paper, we show how standard temporal logics can define n-ary queries over XML documents, thus opening a possibility of using efficient model-checking algorithms [9] in XML querying. We begin with an easy observation that languages capturing binary FO (or MSO) queries can be extended with a simple binary term to capture arbitrary n-ary queries. While some languages for binary FO and MSO are known [25,15], there is an abundance of nice formalisms for unary and Boolean queries, and those logics tend to have very good model-checking properties. Thus, as our main contribution, we present a technique for *combining* temporal logics to obtain languages for n-ary XML queries. To characterize n-ary $\mathcal{L}$ queries, where $\mathcal{L}$ could be FO or MSO (and the result applies to several other logics lying between FO and MSO), one needs:

Ingredients:
- a temporal logic $\mathcal{L}_0$ that captures Boolean $\mathcal{L}$ over words (e.g., LTL for FO, or μ-calculus for MSO);
- a temporal logic $\mathcal{L}_1$ that captures *unary* $\mathcal{L}$ queries over XML trees (quite a few are known [25,32,24,3]: for example, CTL* with the past for FO, or the full μ-calculus for MSO);
- some binary operations on trees, such as the largest common ancestor for two nodes.

Combination mechanism: This comes in the form of XPath's *node tests*: for each formula ψ of $\mathcal{L}_1$, we have a node test $[\psi]$ that becomes an atomic proposition of $\mathcal{L}_0$ and simply checks if ψ is true in a given node.

Let us add a few early comments on binary operations (exact sets of those will be defined later in Section 3). Consider the standard *document order* for XML documents: $s \leq_{\mathrm{d}} s'$ if either s' is a descendant of s, or s occurs ahead of s' as one looks at the string representation of a document:

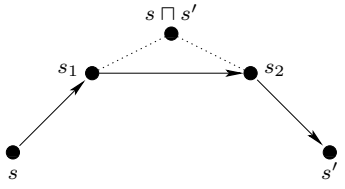

Fig. 1. Document ordering and binary terms

Then a path from s to s' witnessing $s \leq_{\mathrm{d}} s'$ naturally defines two points, s_1 and s_2, where it changes direction. Note that s_1 is the successor of $s \sqcap s'$ in the direction of s, and s_2 is the successor of $s \sqcap s'$ in the direction of s', where $s \sqcap s'$ is the meet (largest common ancestor) of s and s'. This naturally suggests two terms: one of them is the meet $\sqcap$ of two nodes, and the other is the successor of one node in the direction of its descendant. This is the set of terms we use here.

In this paper we look at combined logics that capture n-ary FO and MSO queries. We give their precise definition, prove expressive completeness for n-ary queries, study their model-checking properties, and relate them to XPath queries and XML tree patterns.

2 Notation

Unranked trees as transition systems. A *tree domain* D is a finite prefix-closed subset of $\mathbb{N}^*$ (strings of natural numbers) such that $s \cdot i \in D$ and $j < i$ imply $s \cdot j \in D$. That is, if a node s has n children, they are $s \cdot 0, \ldots, s \cdot (n-1)$. Nodes of trees are labeled by letters from a finite alphabet Σ. A *Σ-tree* is viewed as a transition system

$$T \;=\; (D, \prec_{\mathrm{ch}}, \prec_{\mathrm{sb}}, (P_a)_{a \in \Sigma}),$$

where D is a tree domain, $\prec_{\mathrm{ch}}$ is the child relation ($s \prec_{\mathrm{ch}} s \cdot i$ for all $s, s \cdot i \in D$), $\prec_{\mathrm{sb}}$ is the next-sibling relation ($s \cdot i \prec_{\mathrm{sb}} s \cdot (i+1)$ for all $s \cdot (i+1) \in D$), and P_a's are labeling predicates ($s \in P_a$ iff s is labeled a). We shall write $\prec^*_{\mathrm{ch}}$ and $\prec^*_{\mathrm{sb}}$ for the transitive-reflexive closures of $\prec_{\mathrm{ch}}$ and $\prec_{\mathrm{sb}}$. The root of T is the empty string denoted by ε.

We also use the *document ordering* $s \leq_{\mathrm{d}} s'$ which holds iff s appears before s' if the document is written as a string; i.e., either $s \prec^*_{\mathrm{ch}} s'$, or there exist distinct s_0, s_1, s_2 such that $s_0 \prec_{\mathrm{ch}} s_1 \prec^*_{\mathrm{ch}} s$, $s_0 \prec_{\mathrm{ch}} s_2 \prec^*_{\mathrm{ch}} s'$, and $s_1 \prec^*_{\mathrm{sb}} s_2$ (see Fig. 1).

We shall also view Σ-words as transition systems; the domain of a word w of length n is $\{0, \ldots, n-1\}$, with the successor relation $i \prec i+1$ on it, together with the labeling relations P_a's. We assume, as is common when one deals with temporal logics over words, that each position i can be labeled by more than one symbol from Σ.

FO and MSO over trees. First-order logic (FO) is the closure of atomic formulae under Boolean connectives and first-order quantification $\forall x, \exists x$. MSO in addition allows quantification over sets $\forall X, \exists X$ and new atomic formulae $X(x)$ (or $x \in X$). When we deal with FO which cannot define the transitive closure of a relation, we use $x \prec_{\rm ch}^* y$ and $x \prec_{\rm sb}^* y$, as well as $P_a(x)$, as atomic formulae for trees, and the ordering $<$ as well as $P_a(x)$'s for words. For MSO, one can use either $\prec_{\rm ch}^*$ and $\prec_{\rm sb}^*$, or $\prec_{\rm ch}$ and $\prec_{\rm sb}$, since transitive closure is MSO-definable. We shall only deal with MSO formulae with free first-order variables.

If $\varphi(x_1, \ldots, x_n)$ is an FO or MSO formula with n free variables, it defines an n-ary query on a tree T which produces the set $\{\bar{a} \in D^n \mid T \models \varphi(\bar{a})\}$. We let FO_n (resp., MSO_n) stand for the class of n-ary queries definable in FO (resp., MSO). Queries definable by sentences are *Boolean* queries (they produce yes/no answers) and queries definable in FO_1 and MSO_1 are *unary* queries.

Temporal logics. We shall use standard temporal logics such as LTL, CTL*, and the μ-calculus L_μ, cf. [9]. LTL is interpreted over Σ-words and its syntax is:

$$\varphi, \varphi' \;:=\; a,\ a \in \Sigma \mid\ \varphi \vee \varphi' \ \mid\ \neg\varphi \ \mid\ \mathbf{X}\varphi \ \mid\ \varphi \mathbf{U} \varphi'.$$

(As usual, **X** stands for 'next' and **U** for 'until'.) If we have a word w with n positions $0, \ldots, n-1$ labeled by symbols from Σ, the semantics of $(w, i) \models \varphi$ (that is, φ is satisfied in the ith position) is defined by:

- $(w, i) \models a$ iff i labeled with a;
- $(w, i) \models \varphi \vee \varphi'$ iff $(w, i) \models \varphi$ or $(w, i) \models \varphi'$; $\quad (w, i) \models \neg\varphi$ iff $(w, i) \not\models \varphi$;
- $(w, i) \models \mathbf{X}\varphi$ iff $(w, i+1) \models \varphi$;
- $(w, i) \models \varphi \mathbf{U} \varphi'$ iff there exists $k \geq i$ such that $(w, k) \models \varphi'$ and $(w, j) \models \varphi$ for every $i \leq j < k$.

Each LTL formula φ defines a Boolean query over words, that is, the set of words w such that $(w, 0) \models \varphi$. A theorem by Kamp says that this set of queries is precisely the set of Boolean FO queries over words, i.e. $\mathrm{LTL} = \mathrm{FO}_0$.

For other logics, we need their versions that can refer to the past. $\mathrm{CTL}^\star_{\rm past}$, a version of CTL* with the past operators [21], is given below specifically for unranked trees. The grammars for *state* formulae α (satisfied by a node and thus defining unary queries) and *path* formulae β (satisfied by a path) are:

$$\begin{array}{l} \alpha, \alpha' := a\ (a \in \Sigma) \ \mid\ \neg\alpha \ \mid\ \alpha \vee \alpha' \ \mid\ \mathbf{E}\beta \\ \beta, \beta' := \alpha \ \mid \neg\beta \ \mid\ \beta \vee \beta' \ \mid\ \mathbf{X}_{\rm ch}\beta \ \mid\ \mathbf{X}^-_{\rm ch}\beta \ \mid\ \mathbf{X}_{\rm sb}\beta \ \mid\ \mathbf{X}^-_{\rm sb}\beta \ \mid\ \beta\,\mathbf{U}\,\beta' \ \mid\ \beta\,\mathbf{S}\,\beta' \end{array}$$

Here $\mathbf{X}^-$ is the 'previous' and **S** is the 'since' operator. A path π is a sequence $s_1 s_2 \ldots$ of nodes such that for every j, either $s_j \prec_{\rm ch} s_{j+1}$ or $s_j \prec_{\rm sb} s_{j+1}$. As usual with logics with the past, we require the paths to be maximal: that is, $s_1 = \varepsilon$, and all paths end in a leaf that is also the youngest child of its parent. We define the semantics of path formulae $(T, \pi, \ell) \models \beta$ with respect to a position ℓ in a path (where ℓ is an integer). The truth of state formulae is defined with respect to a node of a tree. The rules are as follows (omitting Boolean connectives):

- $(T, s) \models a$ for $a \in \Sigma$ iff s is labeled a.
- $(T, s) \models \mathbf{E}\beta$ iff there exists a path $\pi = s_1 s_2 \ldots$ and $\ell \geq 1$ such that $s = s_\ell$ and $(T, \pi, \ell) \models \beta$;
- $(T, \pi, \ell) \models \alpha$ iff $(T, s_\ell) \models \alpha$;
- $(T, \pi, \ell) \models \mathbf{X}_{\text{ch}}\beta$ (or $\mathbf{X}^-_{\text{ch}}\beta$) iff $(T, \pi, \ell + 1) \models \beta$ and $s_\ell \prec_{\text{ch}} s_{\ell+1}$ (or if $(T, \pi, \ell - 1) \models \beta$ and $s_{\ell-1} \prec_{\text{ch}} s_\ell$); the rules for $\mathbf{X}_{\text{sb}}$ are analogous.
- $(T, \pi, \ell) \models \beta \mathbf{U} \beta'$ iff there exists $k \geq \ell$ such that $(T, \pi, k) \models \beta'$ and $(T, \pi, j) \models \beta$ whenever $\ell \leq j < k$.
- $(T, \pi, \ell) \models \beta \mathbf{S} \beta'$ iff there exists $k \leq \ell$ such that $(T, \pi, k) \models \beta'$ and $(T, \pi, j) \models \beta$ whenever $k < j \leq \ell$.

The version of the μ-calculus we consider here is the full μ-calculus L_μ^{full} [35] that also allows one to refer to the past. Its formulae are defined as

$$\varphi \ := \top \ \mid \ \bot \ \mid \ a \ \mid \ X \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \Diamond(\prec)\varphi \ \mid \ \mu X.\varphi(X),$$

where $a \in \Sigma$, $\prec$ refers to either $\prec_{\text{ch}}$ or $\prec_{\text{sb}}$, or their inverses: parent ($\prec^-_{\text{ch}}$), and previous sibling ($\prec^-_{\text{sb}}$); X ranges over a collection $\mathcal{V}$ of variables, and in $\mu X.\varphi(X)$, the variable X occurs positively in $\varphi(X)$. The semantics, with respect to a valuation v that associates a set of nodes with each variable, is standard: $\top$ is *true*, $\bot$ is *false*, $\Diamond(\prec)\varphi$ is true in s if φ is true in some s' such that $s \prec s'$, X is true in s iff $s \in v(X)$, and $\mu X.\varphi(X)$ defines the least fixed point of the operator $S \mapsto \{s \mid (T, v[S/X], s) \models \varphi\}$, where $v[S/X]$ refers to a valuation that extends v by assigning S to X. Queries (unary or Boolean) are defined by formulae without free variables.

L_μ over words is defined by using one modality for the successor relation. Over words, L_μ formulae evaluated in the initial position have the power of MSO sentences: $L_\mu = \text{MSO}_0$. For unary queries over unranked trees, we have:

Fact 1. ([3,25,32]) *Over unranked trees,* $\text{CTL}^\star_{\text{past}} = \text{FO}_1$ *and* $L_\mu^{\text{full}} = \text{MSO}_1$.

3 Capturing n-Ary Queries

From binary to n-ary queries. As mentioned in the introduction, there is a simple technique for extending a logic capturing FO_2 or MSO_2 to a logic capturing FO_n or MSO_n. It is already implicit in [33], and we briefly outline it.

Let $\mathcal{Q}_2$ be a collection of binary queries given by formulae $\alpha(x, y)$ with two free variables. We then define $\mathcal{Q}_n$ to be the collection of n-ary queries $\psi(x_1, \ldots, x_n)$ which are Boolean combinations of $\alpha(t, t')$, with $\alpha \in \mathcal{Q}_2$ and t, t' being terms given by the grammar $t, t' := x_i,\ i \in [1, n] \ \mid \ t \sqcap t'$. The meaning of $t \sqcap t'$ is the largest common ancestor of t and t'.

Each $\psi(x_1, \ldots, x_n)$ in $\mathcal{Q}_n$ naturally defines a query that returns a set of n-tuples of nodes in a tree. Using the composition technique, and in particular the composition lemma from [33], one can easily show

Proposition 1. *If* $\mathcal{Q}_2$ *captures* FO_2 *(or* MSO_2*), then* $\mathcal{Q}_n$ *captures* FO_n *(or* MSO_n*, respectively) over unranked trees.*

For example, if $\mathcal{Q}_2$ is the set of binary conditional XPath queries [25], then $\mathcal{Q}_n$ captures FO_n over unranked trees.

However, characterizations of binary FO or MSO over XML trees are not nearly as common as characterizations of Boolean and unary queries (with the notable exceptions of conditional XPath in [25], which captures FO_2, and caterpillars expressions extended with unary MSO tests in [5], which capture MSO_2). Moreover, for Boolean and unary queries much has been invested into efficient query-evaluation and model-checking [9,24,20]. Thus, our goal is to find a way to get a language for n-ary queries out of languages for Boolean and unary queries.

From Boolean and unary queries to n-ary queries. We now show how to characterize n-ary FO and MSO queries by combining temporal logics. In what follows, we assume that:

- $\mathcal{L}_0$ is a temporal logic that, for an arbitrary finite alphabet, captures either Boolean FO or Boolean MSO queries over words over that alphabet;
- $\mathcal{L}_1$ is a logic that, for an arbitrary finite alphabet Σ, captures either unary FO or unary MSO queries over Σ-labeled unranked trees.

We then define a combined logic $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ that will capture FO_n or MSO_n. For now, we use a fixed set of binary relations $(\prec^*_{\mathrm{ch}})$ and $(\prec^*_{\mathrm{sb}})$ and a fixed grammar generating terms, but we shall present alternatives at the end of the section.

Variables. Fix n variables $x_1, \ldots, x_n$. Given a tree T, a valuation v in T is a mapping that assigns to each x_i an element s_i of the domain of T.

Terms. These are given by the grammar:

$$(\mathbb{T}) \qquad t, t' \;:=\; x_i,\ i \in [1,n] \;\mid\; \texttt{root} \;\mid\; t \sqcap t' \;\mid\; \mathrm{succ}(t,t')$$

Each valuation v on the variables extends to a valuation on terms: $v(\texttt{root}) = \varepsilon$, $v(t \sqcap t')$ is the longest common prefix of $v(t)$ and $v(t')$, and $v(\mathrm{succ}(t,t'))$ is defined as the child of $v(t)$ in the direction of $v(t')$. More precisely, if $v(t) \prec^*_{\mathrm{ch}} v(t')$, and s is such that $v(t) \prec_{\mathrm{ch}} s$ and $s \prec^*_{\mathrm{ch}} v(t')$, then $s = v(\mathrm{succ}(t,t'))$. Otherwise we set $v(\mathrm{succ}(t,t')) = v(t)$.

Node tests. We define an alphabet Δ that consists of symbols $[\psi]$ for each formula ψ of $\mathcal{L}_1$ (the notation comes from XPath's node tests, because this is precisely the role of $\mathcal{L}_1$ formulae). Notice that Δ is infinite but in all formulae we shall only use finitely many symbols $[\psi]$ and thus we can restrict ourselves to a finite sub-alphabet used in each particular formula.

Interval formulae. An interval formula is a formula of the form $\chi(t,t')$ where χ is an $\mathcal{L}_0$ formula over a finite subset of Δ, and t, t' are two terms.

The semantics is as follows. Let v be a valuation on x_i's. The *interval* between $s = v(t)$ and $s' = v(t')$ is defined as:

- if $s \prec^*_{\mathrm{ch}} s'$, then the interval is the sequence $s = s_0, s_1, \ldots, s_m = s'$ such that $s_i \prec_{\mathrm{ch}} s_{i+1}$ for each $0 \le i < m$ (and the interval between s' and s is simply listed "backwards": $s' = s_m, \ldots, s_0 = s$);

- if $s \prec_{\mathrm{sb}}^{*} s'$, then it is the sequence $s = s_0, s_1, \ldots, s_m = s'$ such that $s_i \prec_{\mathrm{sb}} s_{i+1}$ for each $0 \leq i < m$ (listed backwards for the interval between s' and s);
- otherwise the interval is just $\{s\}$.

Let $[\psi_1], \ldots, [\psi_r]$ be all the Δ-symbols mentioned in χ. Then the interval between s and s' naturally defines a Δ-word in which s_i is labeled by all $[\psi_p]$'s such that $(T, s_i) \models \psi_p$. Then $(T, v) \models \chi(t, t')$ iff the interval between $v(t)$ and $v(t')$, viewed as a Δ-word, satisfies χ.

$\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ *formulae.* are finally defined as Boolean combinations of the following formulae:

$$t \prec_{\mathrm{ch}}^{*} t', \quad t \prec_{\mathrm{sb}}^{*} t', \quad \chi(t, t'),$$

where t, t' are terms, and $\chi(t, t')$ ranges over interval formulae. Given a valuation v, the semantics of $\chi(t, t')$ has already been defined, and $(T, v) \models t \prec_{\mathrm{ch}}^{*} t'$ (or $t \prec_{\mathrm{sb}}^{*} t'$) iff $v(t) \prec_{\mathrm{ch}}^{*} v(t')$ (or $v(t) \prec_{\mathrm{sb}}^{*} v(t')$, respectively). For $\bar{s} = (s_1, \ldots, s_n)$ and a formula φ we shall write $(T, \bar{s}) \models \varphi$ if $(T, v) \models \varphi$ under the valuation $v(x_i) = s_i, i \leq n$.

Each $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ formula φ then defines an n-ary query which maps a tree T with domain D to $\{\bar{s} \in D^n \mid (T, \bar{s}) \models \varphi\}$.

Theorem 1. *If $\mathcal{L}_0$ captures Boolean* FO *(respectively, Boolean* MSO*) queries over words, and $\mathcal{L}_1$ captures unary* FO *(respectively, unary* MSO*) queries over unranked trees, then the queries definable by $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ are precisely the n-ary* FO *(respectively, n-ary* MSO*) queries over unranked trees.*

The proof of Theorem 1 is based on the composition method, cf. [18,29,33]. We start with the (already mentioned) composition lemma from [33], which was used there to obtain n-ary languages that involved regular or star-free expressions over formulae of FO or MSO in one or two variables, and modify it to eliminate regular expressions and formulae referring to two variables by using temporal logics over words and trees.

Other binary relations and terms. Our choice of terms and binary relations $\prec_{\mathrm{ch}}^{*}$ and $\prec_{\mathrm{sb}}^{*}$ is not the only possible one. In general, if we have a grammar τ defining a set of terms and a collection ρ of binary relations, we can define a logic $\mathcal{I}^n[\tau, \rho](\mathcal{L}_0, \mathcal{L}_1)$ in exactly the same way as $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ except:

1. τ-terms are used in place of the terms defined by the grammar ($\mathbb{T}$), and
2. in Boolean combinations only relations from ρ between terms are used.

Now define a new grammar $\mathbb{T}'$ for terms:

$$(\mathbb{T}') \qquad t, t' \ := \ x_i,\ i \in [1, n] \ \mid \ \texttt{root} \ \mid \ \mathrm{gen\_succ}(t, t'),$$

where $\mathrm{gen\_succ}(s, s')$ is the generalized successor of s in the direction of s'. Its meaning is as follows: look at the path from s to s' which is either a child/parent path (if $s \prec_{\mathrm{ch}}^{*} s'$ or $s' \prec_{\mathrm{ch}}^{*} s$), or next/previous-sibling path (if $s \prec_{\mathrm{sb}}^{*} s'$ or $s' \prec_{\mathrm{sb}}^{*} s$), or the path shown in the Fig. 1 (that witnesses either $s \leq_{\mathrm{d}} s'$ or

$s' \leq_{\mathrm{d}} s$). In the first two cases, gen_succ(s, s') is the successor of s on that path; in the third case, it is the first node where the direction of path changes between child/parent and next/previous sibling. For example, in Fig. 1, gen_succ$(s, s') = s_1$ and gen_succ$(s', s) = s_2$.

Theorem 2. *If $\mathcal{L}_0$ captures Boolean* FO *(respectively, Boolean* MSO*) queries over words, and $\mathcal{L}_1$ captures unary* FO *(respectively, unary* MSO*) queries over unranked trees, then the queries definable by $\mathcal{I}^n[\mathbb{T}', \leq_{\mathrm{d}}](\mathcal{L}_0, \mathcal{L}_1)$ are precisely the n-ary* FO *(respectively, n-ary* MSO*) queries over unranked trees.*

That is, with the new set of terms based on just one binary operation, one can capture all n-ary queries by using only the document ordering.

4 Model-Checking for Combined Logics

We now deal with the complexity of the model-checking problem for $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$, that is, the complexity of checking, for an $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ formula φ, a tree T and an n-tuple $\bar{s}$ of its nodes, whether $(T, \bar{s}) \models \varphi$. (The results will hold for the alternative system of terms and the document order $\leq_{\mathrm{d}}$ as well.)

We first offer a general result that makes some mild assumptions on logics $\mathcal{L}_0$ and $\mathcal{L}_1$. We then consider specific cases of logics $\mathcal{L}_0$ and $\mathcal{L}_1$ so that $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ captures FO_n or MSO_n and provide better complexity bounds.

Let $\mathcal{MC}^{\mathcal{L}} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the complexity of model-checking for a logic $\mathcal{L}$; i.e., given a structure $\mathfrak{M}$ and an $\mathcal{L}$-formula γ, verifying $\mathfrak{M} \models \gamma$ can be done in $O(\mathcal{MC}^{\mathcal{L}}(\|\mathfrak{M}\|, \|\gamma\|))$, where $\|\cdot\|$ is the size of encoding of structures (formulae).

We make three very mild assumptions on model-checking algorithms for $\mathcal{L}_0$ and $\mathcal{L}_1$. First, we assume that formulae are given by their parse-trees; second, that labeling nodes by additional symbols not used in formulae does not change their truth values; and third, that $\mathcal{MC}^{\mathcal{L}}(\cdot, \cdot)$ is a nondecreasing function in both arguments such that $\mathcal{MC}^{\mathcal{L}}(n, m_1) + \mathcal{MC}^{\mathcal{L}}(n, m_2) \leq \mathcal{MC}^{\mathcal{L}}(n, m_1 + m_2)$. All logics considered here – FO, MSO, LTL, CTL*, L_μ, etc. – easily satisfy these properties.

Proposition 2. *If logics $\mathcal{L}_0$ and $\mathcal{L}_1$ satisfy the three properties described above, then the complexity of model-checking for the combined logic $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ is $O\big(\|T\| \cdot \mathcal{MC}^{\mathcal{L}_1}(\|T\|, \|\varphi\|) + \mathcal{MC}^{\mathcal{L}_0}(\|T\|, \|\varphi\|)\big)$.*

These bounds are produced by a naive model-checking algorithm. An $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ formula is a Boolean combination of term comparisons and interval formulae $\chi(t, t')$. To evaluate $\chi(t, t')$ in $(T, \bar{s})$, we define a valuation $v(x_i) = s_i, i \leq n$, and do the following:

1. Compute $v(t)$ and $v(t')$ and the interval between them.
2. For each symbol $[\psi]$ for $\psi \in \mathcal{L}_1$ mentioned in φ, and each s in the interval between $v(t)$ and $v(t')$, mark s with $[\psi]$ if $(T, s) \models \psi$ (by using the model-checking algorithm for $\mathcal{L}_1$).
3. With all elements in the interval marked, use the model-checking algorithm for $\mathcal{L}_0$ to check if χ holds.

The bound easily follows from this and our assumptions on $\mathcal{L}_0$ and $\mathcal{L}_1$.

Even if we assume that $\mathcal{L}_0$ is a logic with very good model-checking complexity (say, $O(\|T\| \cdot \|\varphi\|)$), the bound of Proposition 2 still says that model-checking is quadratic in $\|T\|$, while in XML query processing, generally acceptable complexity is of the form $O(f(\|\varphi\|)\cdot\|T\|)$ for reasonable f [20,24], and ideally $O(\|T\|\cdot\|\varphi\|)$ (see, e.g., [16,25]).

However, the bound can be lowered if we make some assumptions (that will hold in cases of interest) not only on model-checking properties of $\mathcal{L}_1$, but also on the complexity of computing the set $\{s \mid (T,s) \models \psi\}$ for $\mathcal{L}_1$ formulae ψ (that is, on the complexity of unary query evaluation). Assume that there is a function $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(m) + f(k) \leq f(m+k)$ (e.g., $f(m) = c \cdot m^p$ or $f(m) = 2^m$) and a number $\ell > 0$ such that unary query evaluation in $\mathcal{L}_1$ is done in time $f(\|\psi\|) \cdot \|T\|^\ell$. In this case, if an $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ formula φ mentions $[\psi_1], \ldots, [\psi_r]$, we can in time $\sum_i f(\|\psi_i\|) \cdot \|T\|^\ell \leq f(\|\varphi\|) \cdot \|T\|^\ell$ label all nodes in which ψ_i holds with $[\psi_i]$, $1 \leq i \leq r$, and thus check φ in time $O(f(\|\varphi\|) \cdot \|T\|^\ell + \mathcal{MC}^{\mathcal{L}_0}(\|\varphi\|, \|T\|))$. We thus obtain the following:

Theorem 3. *If unary query evaluation in $\mathcal{L}_1$ is done in time $f(\|\psi\|)\cdot\|T\|^\ell$, and the complexity of model-checking for an $\mathcal{L}_0$ formula α on a word w is $g(\|\alpha\|)\cdot\|w\|^p$, then the complexity of model-checking of $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ is*

$$O\big(\max\{f(\|\varphi\|), g(\|\varphi\|)\} \cdot \|T\|^{\max\{\ell,p\}}\big).$$

In particular, if both f and g are linear functions and $\ell = p = 1$, we get an $O(\|\varphi\| \cdot \|T\|)$ model-checking algorithm for $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$.

We now use known results on model-checking over words and trees to obtain good model-checking algorithms for combined logics over unranked trees.

$\mathbf{MSO}_n$ queries. To get a logic $\mathcal{I}^n(\mathcal{L}_0, \mathcal{L}_1)$ that captures MSO_n we need a logic for unary MSO on trees, and a logic for MSO sentences on words. The former is provided by L_μ^{full}, the full μ-calculus [3]. Over trees (in general, acyclic transition systems), L_μ is known to admit $O(\|\varphi\|^2 \cdot \|t\|)$ model-checking complexity [26], but this result does not extend to L_μ^{full} since introduction of the past modalities effectively transforms trees into cyclic transition systems. However, it can be shown by coding query automata [29] that a small fragment of L_μ^{full} suffices to capture MSO_1 over trees. We let $(L_\mu^{\mathrm{full}})^+$ be the fragment of L_μ^{full} that contains no negation (and thus is alternation-free) but is allowed to use additional labels "root", "leaf", "first-sibling", and "last-sibling" [16] with their intuitive meanings.

Lemma 1. *Over unranked trees, $(L_\mu^{\mathrm{full}})^+ = \mathrm{MSO}_1$.*

Unary query evaluation in alternation-free μ-calculus L_μ^+ can be done in linear-time for arbitrary transition systems [10], and hence it is linear-time for $(L_\mu^{\mathrm{full}})^+$ over trees. For words, alternation-free μ-calculus L_μ^0 captures MSO_0 (by coding automata), and again from [10], the complexity of model-checking is linear in both the formula and the word. Combining this with Theorem 3 we get:

Corollary 1. *The logic* $\mathcal{I}^n(L^0_\mu, (L^{\text{full}}_\mu)^+)$ *captures* MSO_n *over unranked trees, and the complexity of* $\mathcal{I}^n(L^0_\mu, (L^{\text{full}}_\mu)^+)$ *model-checking is* $O(\|T\| \cdot \|\varphi\|)$.

FO$_n$ queries. We need logics for Boolean FO queries on words and unary FO on trees. The former is, by Kamp's theorem, LTL, which has linear-time complexity over words.

Among logics used in verification, CTL* with the past is known to capture unary FO over trees (see Fact 1). However, even though it can be embedded in L^{full}_μ, the complexity of CTL* does not match the linear complexity we had for $(L^{\text{full}}_\mu)^+$, being in general $2^{O(\|\varphi\|)} \cdot \|T\|$ (see [12]; also, [4] shows that translation into L^{full}_μ will exhibit exponential blowup).

In fact, we can show that it is highly unlikely that we can get linear time evaluation for $\text{CTL}^\star_{\text{past}}$ over trees. In general, CTL* is known to be PSPACE-complete [34]. Here we show that over trees, the complexity of model-checking is lower, but still intractable, being in the second level of the polynomial hierarchy.

Theorem 4. *The model-checking problem for* $\text{CTL}^\star_{\text{past}}$ *over unranked trees is* Δ^p_2*-complete.*

Proof sketch. The usual algorithm for CTL* model checking combines the state labeling technique for CTL model checking with LTL model checking. Its complexity mainly depends on the complexity of the LTL part. In particular, it runs in polynomial time if we have an oracle for verifying whether a formula $\mathbf{E}\,\varphi$ holds in a state s, where φ is an LTL formula. For unranked trees, it can be proved that the latter problem is NP-complete and, thus, the model-checking problem for $\text{CTL}^\star_{\text{past}}$ is in Δ^p_2. For hardness reduction, we use (as [23] for CTL$^+$) the problem of verifying whether the largest satisfying assignment (interpreted as a binary number) of a propositional formula is even. □

Nonetheless, there is a temporal logic for trees that has the desired linear complexity. The logic, which we call TL^{tree} (for tree temporal logic), was first defined in [32] for the case of trees without a sibling order $\prec_{\text{sb}}$, and further used in XPath investigations [25]. Its syntax is given by:

$$\alpha, \alpha' := \top \mid \bot \mid a\ (a \in \Sigma) \mid \alpha \vee \alpha' \mid \neg\alpha \mid \mathbf{X}_*\alpha \mid \mathbf{X}^-_*\alpha \mid \alpha \mathbf{U}_* \alpha' \mid \alpha \mathbf{S}_* \alpha',$$

where $*$ is either 'ch' (child) or 'sb' (next sibling). We define the semantics with respect to a tree and a node in a tree:

- $(T, s) \models \top$; $(T, s) \not\models \bot$;
- $(T, s) \models a$ iff s is labeled a;
- $(T, s) \models \mathbf{X}_{\text{ch}}\alpha$ if $(T, s \cdot i) \models \alpha$ for some i;
- $(T, s) \models \mathbf{X}^-_{\text{ch}}\alpha$ if $(T, s') \models \alpha$ where s' is the parent of s ($s' \prec_{\text{ch}} s$);
- $(T, s) \models \alpha \mathbf{U}_{\text{ch}} \alpha'$ if there is a node s' such that $s \prec^*_{\text{ch}} s'$, $(T, s') \models \alpha'$, and for all $s'' \neq s'$ satisfying $s \prec^*_{\text{ch}} s'' \prec^*_{\text{ch}} s'$ we have $(T, s'') \models \alpha$.

The semantics of $\mathbf{S}_{\text{ch}}$ is defined by reversing the order in the semantics of $\mathbf{U}_{\text{ch}}$, and the semantics of $\mathbf{X}_{\text{sb}}, \mathbf{X}^-_{\text{sb}}, \mathbf{U}_{\text{sb}}$, and $\mathbf{S}_{\text{sb}}$ is the same by replacing the child relation with the next sibling relation.

Whenever we deal with $\mathrm{TL}^{\mathrm{tree}}$, we assume (for the convenience of translations) that the *weak until* or *unless* operator [9] $\varphi \mathbf{W} \psi \equiv \neg(\neg\psi \mathbf{U} \neg(\varphi \vee \psi))$ is available for each of the until operators. This changes neither expressiveness nor the complexity of model-checking [9].

$\mathrm{TL}^{\mathrm{tree}}$ naturally defines unary queries on trees, and the results in [32] can be extended to show that $\mathrm{TL}^{\mathrm{tree}} = \mathrm{FO}_1$ (see, for instance, [25]). Furthermore, we can show:

Lemma 2. *Unary query evaluation in* $\mathrm{TL}^{\mathrm{tree}}$ *can be done in time* $O(\|T\| \cdot \|\varphi\|)$.

We thus have a logic for FO_n with linear model-checking:

Corollary 2. *The logic* $\mathcal{I}^n(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$ *captures* FO_n *over unranked trees, and the complexity of* $\mathcal{I}^n(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$ *model-checking is* $O(\|T\| \cdot \|\varphi\|)$.

5 Combined Temporal Logics and XML Querying

In this section we present two concrete translations from XML query languages into combined temporal logics. We start with XPath (or, more precisely, CXPath, or conditional XPath [25]). As it captures FO_2, one immediately obtains from Corollary 2 that it can be translated into $\mathcal{I}^2(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$. We present a translation which shows how the main features of combined temporal logics correspond naturally to navigation through XML documents. We then give an example of translating *tree patterns* – a common mechanism for expressing queries for selecting tuples of nodes in XML documents [8,22] – into $\mathcal{I}^n(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$.

From Conditional XPath to $\mathcal{I}^2(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$. Conditional XPath (CXPath) [25] is an extension of the logical core of XPath 1.0 that captures FO_2 queries over XML documents. The language contains basic expressions `step`, path expressions `path`, and node tests `test`, given by the grammar below:

$$\begin{aligned}
&\texttt{step} := \texttt{child} \mid \texttt{parent} \mid \texttt{right} \mid \texttt{left},\\
&\texttt{path} := \texttt{step} \mid ?\texttt{test} \mid (\texttt{step}/?\texttt{test})^+ \mid \texttt{path}/\texttt{path} \mid \texttt{path} \cup \texttt{path},\\
&\texttt{test} := a,\ a \in \Sigma \mid \langle \texttt{path} \rangle \mid \neg\texttt{test} \mid \texttt{test} \vee \texttt{test}.
\end{aligned}$$

Given a tree T, the semantics of a `step` or a `path` expression e is the set $[\![e]\!]_T$ of pairs of nodes, and for a `test` expression e, $[\![e]\!]_T$ is a set of nodes of T. The semantics is defined in Figure 2. Note that '/' is the concatenation of paths, and the $\langle \texttt{path} \rangle$ test corresponds to $\mathbf{E}\beta$ of $\mathrm{CTL}^\star$. We use the notation $\prec^+$ for the transitive closure, that is, $\prec \circ \prec^*$.

Translating FO_2 into CXPath is necessarily non-elementary (which easily follows from the fact that translation from FO to LTL over words is necessarily nonelementary [11]). For the combined logic, we can show:

Theorem 5. *For every CXPath path formula* φ *there exists an equivalent* $\mathcal{I}^2(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$ *formula* φ°. *Moreover,* φ° *can be constructed in single-exponential time.*

$$\begin{array}{rr}
[\![\texttt{child}]\!]_T = \{(s,s') \mid s \prec_{\mathrm{ch}} s'\} & [\![a]\!]_T = \{s \mid s \text{ is labeled } a\} \\
[\![\texttt{parent}]\!]_T = \{(s,s') \mid s' \prec_{\mathrm{ch}} s\} & [\![\texttt{test} \vee \texttt{test}']\!]_T = [\![\texttt{test}]\!]_T \cup [\![\texttt{test}']\!]_T \\
[\![\texttt{right}]\!]_T = \{(s,s') \mid s \prec_{\mathrm{sb}} s'\} & [\![\neg\texttt{test}]\!]_T = D - [\![\texttt{test}]\!]_T \\
[\![\texttt{left}]\!]_T = \{(s,s') \mid s' \prec_{\mathrm{sb}} s\} & [\![\langle\texttt{path}\rangle]\!]_T = \{s \mid \exists s' :\ (s,s') \in [\![\texttt{path}]\!]_T\}
\end{array}$$

$$[\![?\texttt{test}]\!]_T = \{(s,s) \mid s \in [\![\texttt{test}]\!]_T\}$$

$$[\![\texttt{path}/\texttt{path}']\!]_T = [\![\texttt{path}]\!]_T \circ [\![\texttt{path}']\!]_T \qquad [\![\texttt{path} \cup \texttt{path}']\!]_T = [\![\texttt{path}]\!]_T \cup [\![\texttt{path}']\!]_T$$

$$[\![(\texttt{child}/?\texttt{test})^+]\!]_T = \{(s,s') \mid s \prec^+_{\mathrm{ch}} s',\ \text{and } \forall s'' : \big(s \prec^+_{\mathrm{ch}} s'' \prec^*_{\mathrm{ch}} s' \rightarrow s'' \in [\![\texttt{test}]\!]_T\big)\}$$

Fig. 2. The semantics of CXPath

Below we sketch the translation and explain the reason for the exponential blowup (intuitively, it arises from putting CXPath expressions in a certain normal form [25] that fits in nicely with $\mathcal{I}^2(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$).

We start with path expressions. CXPath, as well as XPath 1.0, allows expressions containing any combination of the four axes `child`, `parent`, `right` and `left`, but [25] gave a normal form for paths: namely, every CXPath expression is equivalent to a union of simple paths defined by:

$$\begin{array}{rcl}
\texttt{simple-path} & := & ?\texttt{test} \mid \texttt{dpath} \mid \texttt{upath} \mid \texttt{lpath} \mid \texttt{rpath} \mid \\
& & \texttt{upath}/\texttt{rpath} \mid \texttt{rpath}/\texttt{dpath} \mid \texttt{upath}/\texttt{rpath}/\texttt{dpath} \mid \\
& & \texttt{upath}/\texttt{lpath} \mid \texttt{lpath}/\texttt{dpath} \mid \texttt{upath}/\texttt{lpath}/\texttt{dpath},
\end{array}$$

where `dpath` (down-path) is a concatenation of paths `child`, ?`test` and $(\texttt{child}/?\texttt{test})^+$ that mentions `child` or $(\texttt{child}/?\texttt{test})^+$ at least once; and `upath`, `rpath` and `lpath` (up-, right-, and left-paths) are defined in the same way but replacing `child` by `parent`, `right` and `left`, respectively. Thus, it suffices to provide translations for simple path expressions. As an example we show translations of `dpath` and `upath/rpath/dpath`, as the remaining translations are very similar. For a downpath π, we define an interval formula $\pi^\circ(x_1, x_2)$ such that for every Σ-tree T and a pair of nodes s_1, s_2 in it, $(s_1, s_2) \in [\![\pi]\!]_T$ iff $(T, s_1, s_2) \models x_1 \prec^*_{\mathrm{ch}} x_2 \wedge (\pi^\circ)(x_1, x_2)$:

$$\begin{array}{rcl}
(\texttt{child})^\circ & := & \mathbf{X}\,\neg\mathbf{X}\,\top \\
(?\texttt{test})^\circ & := & [\texttt{test}^\circ] \wedge \neg\mathbf{X}\,\top \\
((\texttt{child}/?\texttt{test})^+)^\circ & := & \mathbf{X}\,\neg(\top\,\mathbf{U}\,\neg[\texttt{test}^\circ]) \\
(\texttt{child}/\texttt{dpath})^\circ & := & \mathbf{X}\,\texttt{dpath}^\circ \\
(?\texttt{test}/\texttt{dpath})^\circ & := & [\texttt{test}^\circ] \wedge \texttt{dpath}^\circ \\
((\texttt{child}/?\texttt{test})^+/\texttt{dpath})^\circ & := & \mathbf{X}\,([\texttt{test}^\circ]\,\mathbf{U}\,(\mathbf{X}^-\texttt{dpath}^\circ)),
\end{array}$$

where $\texttt{test}^\circ$ is the translation of `test` expressions into $\mathrm{TL}^{\mathrm{tree}}$ formulae.

As another example, consider a simple path `upath/rpath/dpath`. Assume that a node s' is reachable from a node s by following this path, as in Fig. 1, where $s_1 = \mathrm{succ}(s \sqcap s', s)$ and $s_2 = \mathrm{succ}(s \sqcap s', s')$. Then `upath/rpath/dpath` is expressed by an $\mathcal{I}^2(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$ formula $\varphi(x_1, x_2)$:

$$(\texttt{upath}^\circ)(x_1, \mathrm{succ}(x_1 \sqcap x_2, x_1)) \wedge \mathrm{succ}(x_1 \sqcap x_2, x_1) \prec^*_{\mathrm{sb}} \mathrm{succ}(x_1 \sqcap x_2, x_2) \wedge$$
$$(\texttt{rpath}^\circ)(\mathrm{succ}(x_1 \sqcap x_2, x_1), \mathrm{succ}(x_1 \sqcap x_2, x_2)) \wedge (\texttt{dpath}^\circ)(\mathrm{succ}(x_1 \sqcap x_2, x_2), x_2).$$

Finally we must deal with node tests which will be translated into $\mathrm{TL}^{\mathrm{tree}}$. Had we used $\mathrm{CTL}^{\star}_{\mathrm{past}}$, the translation would have have been immediate as $\langle \mathtt{path} \rangle$ is simply $\mathbf{E}(\mathtt{path}^{\circ})$. But $\mathrm{TL}^{\mathrm{tree}}$ is more restrictive, and thus our first step is to give an equivalent grammar for CXPath node tests:

$$\begin{array}{rcl} \mathtt{test} & := & a, a \in \Sigma \mid \langle \mathtt{union} \rangle \mid \neg \mathtt{test} \mid \mathtt{test} \vee \mathtt{test} \\ \mathtt{union} & := & \mathtt{concat} \mid \mathtt{union} \cup \mathtt{union} \\ \mathtt{concat} & := & \mathtt{step} \mid ?\mathtt{test} \mid (\mathtt{step}/?\mathtt{test})^{+} \mid \\ & & \mathtt{step}/\mathtt{concat} \mid ?\mathtt{test}/\mathtt{concat} \mid (\mathtt{step}/?\mathtt{test})^{+}/\mathtt{concat} \end{array}$$

Here the semantics is existential: $\mathtt{test}$ is true in s if for some s' it is the case that (s, s') is in the semantics of the corresponding path expression. With the new grammar, the translation (given below for the 'child' axis) is quite straightforward:

$$\begin{array}{rclrcl} a^{\circ} & := & a & \langle \mathtt{union} \rangle^{\circ} & := & \mathtt{union}^{\circ} \\ (\neg \mathtt{test})^{\circ} & := & \neg \mathtt{test}^{\circ} & (\mathtt{test}_1 \vee \mathtt{test}_2)^{\circ} & := & \mathtt{test}_1^{\circ} \vee \mathtt{test}_2^{\circ} \\ (\mathtt{union}_1 \cup \mathtt{union}_2)^{\circ} & := & \mathtt{union}_1^{\circ} \vee \mathtt{union}_2^{\circ} & \mathtt{child}^{\circ} & := & \mathbf{X}_{\mathrm{ch}} \top \\ (?\mathtt{test})^{\circ} & := & \mathtt{test}^{\circ} & ((\mathtt{child}/?\mathtt{test})^{+})^{\circ} & := & \mathbf{X}_{\mathrm{ch}} \mathtt{test}^{\circ} \\ (\mathtt{child}/\mathtt{concat})^{\circ} & := & \mathbf{X}_{\mathrm{ch}} \mathtt{concat}^{\circ} & (?\mathtt{test}/\mathtt{concat})^{\circ} & := & \mathtt{test}^{\circ} \wedge \mathtt{concat}^{\circ} \\ \multicolumn{6}{c}{((\mathtt{child}/?\mathtt{test})^{+}/\mathtt{concat})^{\circ} := \mathbf{X}_{\mathrm{ch}} \neg(\neg \mathtt{concat}^{\circ} \mathbf{W}_{\mathrm{ch}} \neg \mathtt{test}^{\circ})} \end{array}$$

To conclude, we note that the translation of paths into the normal form is exponential [25] and the same is true for the translation for tests; for formulae in normal form, translations into both $\mathrm{TL}^{\mathrm{tree}}$ and $\mathcal{I}^2(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$ are linear, which proves the theorem.

From tree-patterns to $\mathcal{I}^n(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$. Tree-pattern queries are a popular way of navigating in XML documents and retrieving n-ary tuples of nodes [8,22]. Fix an alphabet Σ and n variables $x_1, \ldots, x_n$. Tree-pattern queries use a restricted language for paths (where a ranges over Σ):

$$\begin{array}{rcl} \mathtt{step} & := & \mathtt{self} \mid \mathtt{child} \mid \mathtt{child}^{+} \\ \mathtt{path} & := & \mathtt{step} \mid \mathtt{step}/?a \mid \mathtt{step}/x_i \mid \mathtt{step}/?a/x_i, \quad i \leq n \end{array}$$

Variables retrieve nodes from documents: for example, $\mathtt{self}/x_i$ retrieves the node where the formula is evaluated, and $\mathtt{child}^{+}/?a/x_i$ retrieves all the descendants of a node that are labeled a. Tree-pattern formulae are defined as follows:

$$\varphi \quad := \quad \mathtt{path} \mid \mathtt{path}[\varphi, \ldots, \varphi],$$

with the additional requirement that each variable x_i is mentioned at most once. In a tree-pattern formula, square brackets are used to indicate that a list of paths have a common starting point.

An n-ary tree-pattern formula φ is definable in FO and thus in $\mathcal{I}^n(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$. In fact, one can prove a stronger result:

Proposition 3. *For every tree-pattern formula* φ *one can construct in linear-time an equivalent* $\mathcal{I}^n(\mathrm{LTL}, \mathrm{TL}^{\mathrm{tree}})$ *formula* φ°.

6 Conclusions

Connections between XML querying and temporal logics were discovered recently but familiar logics such as CTL* or the μ-calculus were only suitable for Boolean or unary queries over XML documents. Here we have shown how to combine temporal logics to obtain query languages for selecting arbitrary tuples of nodes from XML trees, that capture the power of FO and MSO querying. The observation that composing monadic queries is sufficient to capture n-ary MSO was also made recently in [13].

One of the main goals of this work is to bring techniques developed in the model-checking community into the field of XML querying, where complexity of query evaluation for languages such as XPath and XQuery is a very recent and active topic of research [17,20]. We have shown that some of the combined logics achieve the best possible complexity of model-checking: linear in both the formula and the document. Two natural extensions of this work are: (1) an experimental evaluation of the combined temporal logics proposed here using existing model-checkers, and (2) further extension of the logics by allowing them to reshape tuples of nodes, thus making them closer to languages such as XQuery.

Acknowledgments. We thank anonymous referees for their comments. Arenas is supported by FONDECYT grant 1050701; Arenas and Barceló by Grant P04-067-F from the Millennium Nucleus Centre for Web Research; Libkin is on leave from the University of Toronto, supported by the European Commission Marie Curie Excellence grant MEXC-CT-2005-024502, EPSRC grant E005039, and a grant from NSERC.

References

1. L. Afanasiev, M. Franceschet, M. Marx, M. de Rijke. CTL model checking for processing simple XPath queries. In *TIME 2004*, pages 117–124.
2. L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. Marx, M. de Rijke: PDL for ordered trees. *J. Appl. Non-Classical Logics* 15 (2005), 115–135.
3. P. Barceló, L. Libkin. Temporal logics over unranked trees. *LICS'05*, pages 31–40.
4. G. Bhat, R. Cleaveland. Efficient model checking via the equational μ-calculus. In *LICS 1996*, pages 304–312.
5. R. Bloem, J. Engelfriet. Monadic second order logic and node relations on graphs and trees. *Struct. in Logic and Comp. Science*, 1997: 144-161.
6. L. Cardelli, G. Ghelli. A query language based on the ambient logic. In *ESOP 2001*, pages 1–22.
7. B. ten Cate. Expressivity of XPath with transitive closure. In *PODS'06*, pages 328–337.
8. Z. Chen, H.V.Jagadish, L. Lakshmanan, S. Paparizos. From tree patterns to generalized tree patterns: on efficient evaluation of XQuery. *VLDB'03*, pages 237–248.
9. E. Clarke, B.-H. Schlingloff. Model Checking. In *Handbook of Automated Reasoning*, Elsevier 2001, pages 1635–1790.
10. R. Cleaveland, B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *CAV'91*, pages 48–58.

11. K. Compton, C.W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *APAL* 48 (1990), 1–79.
12. E. A. Emerson, C.-L. Lei. Modalities for model checking: branching time logic strikes back. *Sci. Comput. Program.* 8 (1987), 275–306.
13. E. Filiot, J. Niehren, J-M. Talbot, S. Tison. Composing monadic queries in trees. In *PLAN-X* 2006: 61-70.
14. M. Frick, M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *LICS 2002*, 215–224.
15. E. Goris, M. Marx. Looping caterpillars. In *LICS 2005*, pages 51–60.
16. G. Gottlob, C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM* 51 (2004), 74–113.
17. G. Gottlob, C. Koch, R. Pichler, L. Segoufin. The complexity of XPath query evaluation and XML typing. *J. ACM* 52 (2005), 284–335.
18. T. Hafer, W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *ICALP'87*, pages 269–279.
19. N. Klarlund, Th. Schwentick, D. Suciu. XML: model, schemas, types, logics, and queries. In *Logics for Emerging Applications of Databases*, Springer 2003.
20. C. Koch. Processing queries on tree-structured data efficiently. In *PODS'06*, pages 213–224.
21. O. Kupferman, A. Pnueli. Once and for all. In *LICS'95*, pages 25–35.
22. L. Lakshmanan, G. Ramesh, H. Wang and Z. Zhao. On testing satisfiability of tree pattern queries. In *VLDB'04*, pages 120–131.
23. F. Laroussinie, N. Markey and Ph. Schnoebelen. Model checking CTL^+ and FCTL is hard. In *FoSSaCS'01*, pages 318–331.
24. L. Libkin. Logics for unranked trees: an overview. In *ICALP'05*, pages 35–50.
25. M. Marx. Conditional XPath. *ACM TODS* 30(4) (2005).
26. R. Mateescu. Local model-checking of modal mu-calculus on acyclic labeled transition systems. In *TACAS'02*, pages 281–295.
27. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM* 51(1): 2–45 (2004).
28. F. Neven. Automata, logic, and XML. In *CSL 2002*, pages 2–26.
29. F. Neven, Th. Schwentick. Query automata over finite trees. *TCS*, 275 (2002), 633–674.
30. D. Niwinski. Fixed points vs. infinite generation. In *LICS 1988*, pages 402–409.
31. L. Planque, J. Niehren, J.M. Talbot, S. Tison. N-ary queries by tree automata. In *DBPL'05*, pages 217–231.
32. B.-H. Schlingloff. Expressive completeness of temporal logic of trees. *Journal of Applied Non-Classical Logics* 2 (1992), 157–180.
33. Th. Schwentick. On diving in trees. In *MFCS'00*, pages 660-669.
34. A. P. Sistla, E. Clarke. The complexity of propositional linear temporal logics. *J. ACM* 32 (1985), 733–749.
35. M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP'98*, pages 628–641.
36. M. Y. Vardi. Model checking for database theoreticians. In *ICDT'05*, pages 1–16.