Query Reasoning on Trees with Types, Interleaving, and Counting

Everardo Bárcenas,[1] **Pierre Genevès,**[2] **Nabil Layaïda,**[1] **and Alan Schmitt**[1]
[1] INRIA, France
[2] CNRS, France

Abstract

A major challenge of query language design is the combination of expressivity with effective static analyses such as query containment. In the setting of XML, documents are seen as finite trees, whose structure may additionally be constrained by type constraints such as those described by an XML schema. We consider the problem of query containment in the presence of type constraints for a class of regular path queries extended with counting and interleaving operators. The counting operator restricts the number of occurrences of children nodes satisfying a given logical property. The interleaving operator provides a succinct notation for describing the absence of order between nodes satisfying a logical property. We provide a logic-based framework supporting these operators, which can be used to solve common query reasoning problems such as satisfiability and containment of queries in exponential time.

1 Introduction

XML is a key technology for describing and exchanging a wide variety of data on the Web. Analyses of XML documents with complex constructions such as queries and types has recently been studied [Genevès *et al.*, 2007; Calvanese *et al.*, 2010]. XML documents can be seen as finite labelled trees, queries are expressed by regular paths (XPath), and XML types (XML Schema, DTDs, RelaxNG) are written in terms of regular tree types. The expressive power of regular queries corresponds to first order logic with two variables FO^2 and regular types to monadic second order logic MSO. Modal logics recently became popular in the context of XML due to their high expressiveness and nice computational properties [Genevès *et al.*, 2007; Calvanese *et al.*, 2010]. We present in this work an analysis framework, based on modal logic, for regular path queries and types equipped with counting and interleaving.

Motivations The XPath query language is used to select nodes in documents. The entire XPath language contains some constructs for counting and others for performing equality tests on values. However, it is well-known that the general combination of these features makes query reasoning undecidable.

In the present work, paths are expressed in terms of a modal tree logic equipped with converse and recursive navigation, graded modalities (counting), and nominals. This logic is known to be undecidable when interpreted over graphs [Bonatti *et al.*, 2008]. We show that the logic is decidable in exponential time when interpreted over finite trees.

Regular tree types can be seen as the arborescent version of regular expressions over strings. A tree type specifies a set of trees (called valid XML documents). The interleave operator of XML schema allows one to describe a complex combination of ordered and unordered contents in unranked trees: $e_1 \& e_2 \& \ldots \& e_n$ denotes the set of trees matching the type expressions e_1 to e_n regardless of their order.

Contributions and Outline We first define an extension of the navigational core of the XPath language. The extension consists in adding constructs for restricting the cardinality of children paths. We also extend regular tree types with constructs for interleaving and counting. We proceed to develop a tree logic equipped with recursion, backward navigation, counting, and nominals. We also show that the logic can capture regular queries and types in a compact manner. We describe a satisfiability algorithm for the logic with exponential time complexity resulting in an efficient reasoning framework for regular paths and types with counting and interleaving.

2 Regular Path Queries

Navigation paths constitute the core of the XPath standard query language. In their simplest form, XPath queries look like directory navigation paths. For example, from a given context node, the expression ch :: a/ac :: b navigates to the *children* (ch) labelled with a, and from there it selects the *ancestor* (ac) nodes labelled with b. Qualifier (filter) expressions may also be used. For instance the expression ch :: a[ac :: b], selects the a *children* nodes with at least one *ancestor* node b.

We extend path queries with numerical constraints on children nodes. We support for instance expressions like ac :: a[ch::$b \leq 5$]. This expression selects a *ancestors* with less than 5 *children* labelled b. Formally, the syntax of a path query P is defined as follows.

$$
\begin{aligned}
a &:= \text{ch} \mid \text{fs} \mid \text{pn} \mid \text{ps} \mid \text{ds} \mid \text{ac} \\
P' &:= \top \mid a \mid p \mid a :: p \mid P'/P' \mid P'[Q] \\
Q &:= P \mid \neg Q \mid Q \vee Q \mid C \\
C &:= \text{ch} :: p > k \mid \text{ch} > k \mid \text{ch} :: p[Q] > k \mid \text{ch}[Q] > k \\
P &:= P' \mid /P' \mid P \cup P \mid P \cap P \mid P \setminus P
\end{aligned}
$$

Relations between nodes are expressed with axes: ch stands for *children*, fs for *following siblings*, pn for *parent*, ps for *previous sibling*, ds for *descendants*, and ac for *ancestors*. In its basic form, a path P consists of a step $a :: p$, which selects the nodes reachable by the axis a and labelled with p. Steps without restrictions on the labeling are written a. The label of the current node is simply written p. The composition of paths is written P_1/P_2. A qualified path $P[Q]$ selects the nodes in the path P that satisfy the boolean condition over paths denoted by the qualifier Q. In addition, the number of children satisfying a qualifier can be bounded: $\text{ch} :: p > k$ is true if and only if the number of p children is *greater than* the natural number k; $\neg(\text{ch} :: p > k)$ holds iff the number of p children is *less or equal than* k. Numerical restrictions can also be imposed on children nodes regardless of their label: $\text{ch} > k$. Finally, $/P$ denotes the nodes that are reachable by P starting from the root. Intersection and union of paths are interpreted as set intersection and union. $P_1 \setminus P_2$ denotes the nodes in P_1 that are not in P_2. We refer the reader to [Genevès *et al.*, 2007] for a formal semantics of XPath node-selection.

3 Regular Tree Types

Regular tree types describe structural constraints for unranked trees (XML documents). They encompass most of the features found in schemas such as DTDs, XML Schemas, and RelaxNGs except interleaving and counting. Interleaving is however a common construct in formal languages. The expression $e_1 \& e_2$ *interleaves* the words (or trees) denoted by e_1 and e_2 in every possible way. It is inductively defined:

$$
\begin{aligned}
e \& \epsilon &= \epsilon \& e = e \\
p_1 e_1 \& p_2 e_2 &= p_1(e_1 \& p_2 e_2) + p_2(p_1 e_1 \& e_2)
\end{aligned}
$$

where ϵ is denotes the empty word (or tree), p_i are propositions and e_i and e can be any expression. $+$ stands for disjunction (alternation) and concatenation is expressed as usual.

The interleave operator is in general exponentially more succinct that its full expansion [Gelade, 2010]. For instance:

$$
\begin{aligned}
p_1 p_2 \,\&\, p_3 p_4 = & p_1 p_2 p_3 p_4 + p_1 p_3 p_2 p_4 + p_1 p_3 p_4 p_2 + \\
& p_3 p_1 p_2 p_4 + p_3 p_1 p_4 p_2 + p_3 p_4 p_1 p_2
\end{aligned}
$$

We extend regular tree types with operators for interleaving and counting on children nodes. Specifically, the syntax of extended regular tree types is defined as follows:

$$
\begin{aligned}
\mathcal{T} := & \epsilon \mid x \mid p\,[\mathcal{T}] \mid \mathcal{T} \cdot \mathcal{T} \mid \mathcal{T} + \mathcal{T} \mid \text{let } \overline{x}.\overline{\mathcal{T}} \text{ in } \mathcal{T} \\
& \mid p\,[\mathcal{T}'_1 \& \ldots \& \mathcal{T}'_n] \mid p\left[\mathcal{T}^{>k}\right] \mid p\left[\mathcal{T}^{\leq k}\right]
\end{aligned}
$$

Disjunction-free types $\mathcal{T}'$ are as follows, where the valuation of x' is a disjunction-free type $\mathcal{T}'$.

$$
\begin{aligned}
\mathcal{T}' := & x' \mid p\,[\mathcal{T}] \mid \mathcal{T}' \cdot \mathcal{T}' \\
& \mid p\,[\mathcal{T}'_1 \& \ldots \& \mathcal{T}'_n] \mid p\left[\mathcal{T}^{>k}\right] \mid p\left[\mathcal{T}^{\leq k}\right]
\end{aligned}
$$

We refer the reader to [Hosoya *et al.*, 2005] for a denotational semantics of regular tree types. We use the following common syntactic sugar through the paper: $\mathcal{T}^? = \epsilon + \mathcal{T}$, $\mathcal{T}^\star = \text{let } x.\mathcal{T} \text{ in } \mathcal{T}x + \epsilon$, and $\mathcal{T}^+ = \mathcal{T}\mathcal{T}^\star$.

Notice that expressions like $p\left[p_1\,[\epsilon]^\star \,\&\, p_2\,[\epsilon]\right]$ are not allowed. This is because the Kleene star is not supported at top level of interleaving. Notice however that recursion can be used if it is not occurring at top level: $p\left[p_1\left[p_3\,[\epsilon]^\star\right] \&\, p_2\,[\epsilon]\right]$.

In XML schemas, counting constructors are used to impose bounds on the number of occurrences of nodes matching regular expressions. The semantics is analog to counting on regular path queries. For instance, $p_0\left[p_1\,[\epsilon]^{>5}\right]$ denotes trees whose root is labelled with p_0 has at least 6 children leaves labelled with p_1. Notice that the children are not required to be immediate siblings, in constrast with other forms of counting in formal languages [Gelade, 2010].

4 XML Logic

We now present a unified reasoning framework for regular path queries and tree types, as a modal tree logic equipped with recursion, backward navigation, counting, and nominals. We then show how to use the logic to efficiently encode regular queries and types with counting and interleaving.

There is a well known bijective encoding between unranked trees and binary trees: one edge is interpreted as the first child relation, and a second edge is used for the next sibling relation. We therefore consider binary trees as models for our logic, without loss of generality.

The syntax of logical formulas, in negation normal form, is defined as follows:

$$
\begin{aligned}
\phi := & p \mid \top \mid x \mid \neg p \mid \neg\top \mid \neg\langle m\rangle\top \mid \phi \vee \phi \mid \phi \wedge \phi \mid \\
& \langle m\rangle\phi \mid \mu x.\phi \mid \phi^{\langle k\rangle} \mid \phi^{[k]}
\end{aligned}
$$

Formulas are interpreted as sets of tree nodes. We follow the traditional transition systems semantics of formulas in the Kripke style but interpreted over finite tree models. A Kripke tree K is a triple $(\mathcal{N}, \mathcal{R}, L)$, where $\mathcal{N}$ is a finite set of nodes, $\mathcal{R} : \mathcal{N} \times \mathcal{M} \mapsto \mathcal{N}$ is transition relation of nodes and modalities ($\mathcal{M}$ is the set of modalities) forming a tree structure, and L is a function that labels nodes with propositions. Informally, $\top$ denotes the full set of nodes, propositions p label nodes, and modalities m denote the transitions between nodes, i.e., $\downarrow$ (*first child*), $\rightarrow$ (*next sibling*), $\uparrow$ (*parent*), and $\leftarrow$ (*previous sibling*). A formula $\langle m\rangle\phi$ denotes the nodes that can access at least one node through m such that ϕ holds. Conjunction and disjunction are interpreted as set intersection and union.

A fixpoint μ is interpreted as a least fixpoint and is used for finite recursion. The formula $\phi = p_1 \wedge \mu x.\langle\uparrow\rangle(p_0 \vee x) \vee \langle\leftarrow\rangle x$ denotes the p_1 *descendants* of p_0. We assume that variables cannot be free and that they can only occur under the scope of a modality or a graded operator. In order to prove correctness of our satisfiability algorithm, we disallow variables to occur under the scope of both a modality and its converse.

A graded formula $\phi^{\langle k\rangle}$ denotes the set of nodes which have *at least* $k+1$ children where ϕ holds. $\phi^{[k]}$ stands for the

nodes where ϕ holds in *all but at most* k children nodes. For example, $p_2^{\langle 100\rangle}$ holds at nodes with at least 101 children p_2.

Negation normal form of formulas can be obtained from usual DeMorgan's rules and $\neg\langle m\rangle\phi = \langle m\rangle\neg\phi \vee \neg\langle m\rangle\top$, $\neg\mu x.\phi = \mu x.\neg\phi\,[^x/_{\neg x}]$, $\neg(\phi^{\langle k\rangle}) = (\neg\phi)^{[k]}$, and $\neg(\phi^{[k]}) = (\neg\phi)^{\langle k\rangle}$. In the sequel, we thus implicitly consider $\neg\phi$ in negation normal form for any ϕ. We write $\phi^{=k}$ instead of $\phi^{\langle k-1\rangle} \wedge (\neg\phi)^{[k]}$ for $k > 0$.

[Bonatti *et al.*, 2008] presented a way to encode graded formulas into plain μ-calculus (without counting): $p^{\langle 2\rangle}$ is equivalent to $\langle\rightarrow\rangle\mu x_1.(p \wedge \langle\rightarrow\rangle\mu x_2.(p \wedge \langle\rightarrow\rangle\mu x_3.p \vee \langle\rightarrow\rangle x_3) \vee \langle\rightarrow\rangle x_2)\vee\langle\rightarrow\rangle x_1$. However, this encoding produces exponentially larger expressions. Since satisfiability of μ-calculus without graded modalities is already exponential time complete, such an encoding of graded modalities yields a doubly exponential time decision procedure.

A nominal can be easily expressed by a constant-size logical formula that explicitly enforces a proposition to occur exactly once, as noticed in [Calvanese *et al.*, 2010].

4.1 Query Translation

We show how regular path queries with counting operators can be translated into logical formulas. First, notice that all axes in regular path queries can be expressed by formulas. The translation function F takes an axis and some context X as input and returns the corresponding logical formula:

$$\begin{aligned}
F(\text{ch}, X) &= \mu x.\langle\uparrow\rangle X \vee \langle\leftarrow\rangle x\\
F(\text{fs}, X) &= \mu x.\langle\leftarrow\rangle X \vee \langle\leftarrow\rangle x\\
F(\text{pn}, X) &= \langle\downarrow\rangle\mu x.X \vee \langle\rightarrow\rangle x\\
F(\text{ps}, X) &= \mu x.\langle\rightarrow\rangle X \vee \langle\rightarrow\rangle x\\
F(\text{ds}, X) &= \mu x.\langle\uparrow\rangle(X \vee x) \vee \langle\leftarrow\rangle x\\
F(\text{ac}, X) &= \langle\downarrow\rangle\mu x.X \vee \langle\downarrow\rangle x \vee \langle\rightarrow\rangle x
\end{aligned}$$

Steps are thus translated as the following conjunction: $F(a::p, X) = F(a, X) \wedge p$. For instance, given a context X, the query $\text{ac} :: p$ is translated into the logic as the following formula: $p \wedge \langle\downarrow\rangle\mu x.X \vee \langle\downarrow\rangle x \vee \langle\rightarrow\rangle x$.

For multi-step queries, the translation function is composed: $F(P_1/P_2, X) = F(P_2, F(P_1, X))$.

Qualified query paths are expressed by the following conjunction: $F(P[Q], X) = F(P, X) \wedge F'(Q, \top)$ where F' is defined below. Notice that the context is not duplicated in this formula. Boolean combinations of qualifiers are trivially translated as follows: $F'(\neg Q, X) = \neg F'(Q, X)$ and $F'(Q_1 \vee Q_2, X) = F'(Q_1, X) \vee F'(Q_2, X)$.

Counting operators are translated as follows:

$$\begin{aligned}
F'(\text{ch}::p > k, X) &= (p \wedge X)^{\langle k\rangle}\\
F'(\text{ch} > k, X) &= X^{\langle k\rangle}\\
F'(\text{ch}::p[Q] > k, X) &= F(p[Q], X))^{\langle k\rangle}\\
F'(\text{ch}[Q] > k, X) &= F(\top[Q], X))^{\langle k\rangle}
\end{aligned}$$

For instance, we translate $\text{ac} :: p_1[\text{ch} :: p_2 > 5]$ for a given context X as: $(p_1 \wedge \langle\downarrow\rangle\mu x.X \vee \langle\downarrow\rangle x \vee \langle\rightarrow\rangle x) \wedge p_2^{\langle 5\rangle}$.

Paths occurring inside qualifiers cannot be translated in the same manner than paths occurring outside. Focus on the selected nodes remains outside the qualifiers:

$$\begin{aligned}
F'(P_1/P_2, X) &= F'(P_1, F'(P_2, X))\\
F'(P[Q], X) &= F'(P, X \wedge F(Q, \top))\\
F'(a::p, X) &= F(\overline{a}, X \wedge p)\\
F'(a) &= F(\overline{a})
\end{aligned}$$

where $\overline{\overline{a}} = a$, $\overline{\text{ch}} = \text{pn}$, $\overline{\text{fs}} = \text{ps}$ and $\overline{\text{ds}} = \text{ac}$. For instance, the query $\text{ch}::p_1[\text{ch}::p_2]$ is translated, given a context X, as: $(p_1 \wedge \mu x.\langle\uparrow\rangle X \vee \langle\leftarrow\rangle x) \wedge \langle\downarrow\rangle\mu x.(p_2 \wedge \top) \vee \langle\rightarrow\rangle x$.

We now conclude the translation of queries: $F(/P, X) = F(P, X \wedge \neg\langle\uparrow\rangle\top \wedge \neg\langle\leftarrow\rangle\top)$, $F(P_1 \cap P_2, X) = F(P_1, X) \wedge F(P_2, X)$, $F(P_1 \cup P_2, X) = F(P_1, X) \vee F(P_2, X)$ and $F(P_1 \setminus P_2, X) = F(P_1, X) \wedge \neg F(P_2, X)$.

Theorem 4.1. *Regular path queries with counting operators translate into logical formulas of linear size.*

4.2 Type Translation

Regular tree types have the same expressive power than MSO and can be linearly translated (without counting and interleaving) into μ-calculus (see for instance [Genevès *et al.*, 2007]). We now show how graded modalities can be used to encode the counting and interleave operators. Given a linear translation of regular tree types without counting and interleaving operators F, we translate inductively the counting operator:

$$\begin{aligned}
F(p\,[\mathcal{T}^{>k}]) &= p \wedge F(\mathcal{T})^{\langle k\rangle} \wedge \neg\langle\rightarrow\rangle\top\\
F(p\,[\mathcal{T}^{\leq k}]) &= p \wedge (\neg F(\mathcal{T}))^{[k]} \wedge \neg\langle\rightarrow\rangle\top
\end{aligned}$$

Graded formulas are now used to encode interleaving:

$$\begin{aligned}
F(p\,[\mathcal{T}_1 \& \ldots \& \mathcal{T}_n]) = p \wedge \bigwedge_{i=1}^{n} (F(\mathcal{T}_i) \wedge @\text{n}_i)^{=1}\\
\wedge\, \top^{=k} \wedge \neg\langle\rightarrow\rangle\top
\end{aligned}$$

where k is the sum of the number of propositions of each $\mathcal{T}_i$. For instance, $p_0\,[p_1\,[\epsilon]\,\&p_2\,[\epsilon]]$ is translated as $p_0 \wedge (\phi \wedge @\text{n}_1)^{=1} \wedge (\psi \wedge @\text{n}_2)^{=1} \wedge \top^{=2} \wedge \neg\langle\rightarrow\rangle\top$, where $\phi = p_1 \wedge \neg\langle\downarrow\rangle\top$ and $\psi = p_2 \wedge \neg\langle\downarrow\rangle\top$. The number of occurrences of p_1 and p_2 is set to exactly one, and there is no restriction in the occurrence order of the labels.

A particular attention is paid to the occurrence order of labels in concatenations occurring at top level of the interleave operator. We translate these concatenations as follows:

$$F(\mathcal{T}_1\mathcal{T}_2) = F(\mathcal{T}_1) \wedge \langle\rightarrow\rangle\mu x.F(\mathcal{T}_2) \vee \langle\rightarrow\rangle x$$

For instance, $p_0\,[p_1\,[\epsilon]\,\&(p_2\,[\epsilon]\,p_3\,[\epsilon])]$ is translated as $p_0 \wedge (\phi \wedge @\text{n}_1)^{=1} \wedge (\xi \wedge @\text{n}_2)^{=1} \wedge \top^{=3} \wedge \neg\langle\rightarrow\rangle\top$, with ϕ as defined above, and $\xi = p_2 \wedge \neg\langle\downarrow\rangle\top \wedge \langle\rightarrow\rangle\mu x.(p_3 \wedge \neg\langle\downarrow\rangle\top) \vee \langle\rightarrow\rangle x$. The formula ξ enforces p_3 to be a following sibling of p_2.

Nominals are used to distinguish identical labels. For example, $p_0\,[p_1\,[\epsilon]\,\&(p_2\,[\epsilon]\,p_1\,[\epsilon])]$ is translated as $p_0 \wedge (\phi \wedge @\text{n}_1)^{=1} \wedge (\xi' \wedge @\text{n}_2)^{=1} \wedge \top^{=3} \wedge \neg\langle\rightarrow\rangle\top$, where $\xi' = p_2 \wedge \neg\langle\downarrow\rangle\top \wedge \langle\rightarrow\rangle\mu x.(p_1 \wedge \neg\langle\downarrow\rangle\top) \vee \langle\rightarrow\rangle x$. As the label p_1 occurs in both ϕ and ξ', we distinguish nodes with the same label with a fresh nominal.

Theorem 4.2. *Regular tree types with counting and interleaving translate into logical formulas whose sizes are at most polynomial.*

Since the logic is closed under negation, we are able to formulate several typical query reasoning problems into the logic. For instance, consider two queries P_1 and P_2, and two types $\mathcal{T}_1$ and $\mathcal{T}_2$. If the formula $F(P_1, F(\mathcal{T}_1)) \wedge \neg F(P_2, F(\mathcal{T}_2))$ is unsatisfiable, then all nodes selected by P_1 under type constraint $\mathcal{T}_1$ are also selected by P_2 under $\mathcal{T}_2$. For solving such reasoning problems, we now present a satisfiability-testing algorithm for formulas of the logic.

5 Satisfiability

We introduce a satisfiability algorithm à la Fischer-Ladner for the logic, in the style of [Genevès *et al.*, 2007; Demri and Lugiez, 2010]. This is done in two steps: we first define the nodes of tree models, then we show how the algorithm builds the candidate tree models. We also show that the algorithm is correct and that its complexity is exponential time.

5.1 Nodes and Trees

During the construction of candidate trees, we use counters of children nodes in order to test graded formulas. We assume that numbers occurring in graded formulas are coded in binary. We then define the counters in a binary form. A binary number is written as a boolean combination of atomic propositions. For instance, 1 is written $p_0 \wedge \neg p_1 \wedge \ldots \wedge \neg p_n$, the number 5 (101 in binary) is written $p_3 \wedge p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n$, for some bound n.

We define the following constant which is a bound for the counters. Given a formula ϕ, we define $\mathcal{C} = n(k+1)$, where n is the number of subformulas $\psi^{\langle k' \rangle}$ occurring in ϕ, and k is the greatest number occurring in the subformulas $\psi^{\langle k' \rangle}$.

Intuitively, the nodes of a given formula ϕ are defined as sets of subformulas of ϕ, such that a formula holds at a node iff such formula is contained in the node. In order to test graded formulas, we use counters to verify the number of children nodes. A bound for the sizes of the counters (number of children) in now introduced.

Theorem 5.1. *If a formula ϕ is satisfiable, then there is a model of ϕ where each node has at most $\mathcal{C}$ children.*

For the proof, we follow the construction of [Bonatti *et al.*, 2008], where the bound is proven for possibly infinite models.

The Fischer-Ladner closure of a given formula is the set of its subformulas and their negation normal form, such that the fixpoints are expanded once, and a counter for each graded subformula is considered. Before defining the closure, consider the following relation for $i = 1, 2$:

$$R^{\text{fl}}(\phi_1 \wedge \phi_2, \phi_i) \quad R^{\text{fl}}(\phi_1 \vee \phi_2, \phi_i) \quad R^{\text{fl}}(\langle m \rangle \psi, \psi)$$
$$R^{\text{fl}}\left(\mu x.\psi, \psi\left[{}^{\mu x.\psi}/_{x}\right]\right) \quad R^{\text{fl}}(\psi, \neg\psi)$$

For graded formulas, we also consider the fixpoint formulas that verify the existence of children nodes, and counters for each graded subformula:

$$R^{\text{fl}}(\psi^{\langle k \rangle}, \langle\downarrow\rangle \mu x.\psi \vee \langle\rightarrow\rangle x) \quad R^{\text{fl}}(\psi^{\langle k \rangle}, \psi^{k'})$$
$$R^{\text{fl}}(\psi^{[k]}, \langle\downarrow\rangle \mu x.\psi \vee \langle\rightarrow\rangle x) \quad R^{\text{fl}}(\psi^{[k]}, \psi^{k'}) \quad R^{\text{fl}}(\psi^{[k]}, \neg\psi^{k'})$$

where $k' = 0, \ldots, \mathcal{C}$. $\psi^{k'}$ is a corresponding counter for each graded formula, that is, it is the number k' coded in binary, as defined above with a set of fresh propositions.

For a given formula ϕ, the Fischer-Ladner closure $\text{FL}(\phi)$ is defined as $\text{FL}(\phi) = \text{FL}(\phi)_n$, such that k is the smallest integer satisfying $\text{FL}(\phi)_{n+1} = \text{FL}(\phi)_n$, where:

$$\text{FL}(\phi)_0 = \{\phi\}$$
$$\text{FL}(\phi)_{i+1} = \text{FL}(\phi)_i \cup \{\psi \mid R^{\text{fl}}(\xi, \psi), \xi \in \text{FL}(\phi)_i\}.$$

We are now ready to define the lean set, which contains the formulas forming the tree nodes. Consider a formula ϕ and a proposition σ not occurring in ϕ. Given that $i = 1, \ldots, \mathcal{C}$, we define the lean set of ϕ as follows:

$$lean(\phi) = \{p, \langle m \rangle \psi, \psi^{\langle k \rangle}, \psi^{[k]}, \psi^{k} \in \text{FL}(\phi)\} \cup \{\sigma, \langle m \rangle \top, \top^{i}\}$$

We now show that the size of the lean is not significantly increased w.r.t. to the formula. The size of a formula is defined inductively in the standard way.

Lemma 5.1. *Given a formula ϕ, the cardinality of $lean(\phi)$ is at most polynomial w.r.t. the size of ϕ.*

A ϕ-node of a formula ϕ, written n^{ϕ}, is defined as a subset of $lean(\phi)$, such that: at least one proposition (different than the ones used for counters) is present; at least one counter is present; when $\langle m \rangle \psi$ occurs, also does $\langle m \rangle \top$; and $\langle\uparrow\rangle \top$ and $\langle\leftarrow\rangle \top$ cannot occur simultaneously. N^{ϕ} denotes the set of ϕ-nodes. In the sequel, we call a ϕ-node simply a node.

We define an entailment relation between nodes and formulas (in negation normal form) as follows:

$$\frac{}{n \vdash \top} \qquad \frac{\phi \in n}{n \vdash \phi} \qquad \frac{\phi \notin n}{n \vdash \neg\phi} \qquad \frac{n \vdash \phi \quad n \vdash \psi}{n \vdash \phi \wedge \psi}$$
$$\frac{n \vdash \phi}{n \vdash \phi \vee \psi} \qquad \frac{n \vdash \psi}{n \vdash \phi \vee \psi} \qquad \frac{n \vdash \phi\left[{}^{\mu x.\phi}/_{x}\right]}{n \vdash \mu x.\phi}$$

A tree is inductively defined as: the empty set $\emptyset$; or a triple (n, X_1, X_2), where n is a node and X_1 and X_2 are trees. The root of a non-empty tree (n, X_1, X_2) is n.

The notion of entailment is extended to trees: a tree X entails a formula ϕ, written $X \Vdash \phi$, iff there is a node n in X such that $n \vdash \phi$ and there are no pending modalities, that is, formulas $\langle\uparrow\rangle\psi$ or $\langle\leftarrow\rangle\psi$ are not contained in the root of X. We write $X \nVdash \phi$ when X does not entail ϕ.

5.2 The Algorithm

The algorithm builds consistent candidate trees in a bottom-up manner. At each step, it checks whether one of the newly built trees satisfies the formula. For a given formula ϕ, the algorithm is as follows:

$Y \leftarrow N^{\phi}$
$X \leftarrow Leaves(Y)$
$X_0 \leftarrow \emptyset$
while $X \nVdash \phi$ or $X \neq X_0$ **do**
  $X_0 \leftarrow X$
  $(X, Y) \leftarrow Update(X, Y)$
end while
if $X \Vdash \phi$ **then**
  return ϕ is satisfiable

end if

return ϕ is not satisfiable

Initially tested nodes are leaves. Given a set of nodes X, a node is a leaf $n \in Leaves(X)$ iff $n \in X$, formulas $\langle\downarrow\rangle\psi$ or $\langle\downarrow\rangle\psi$ do not occur in n, and the counters are initialized, that is, for every $\psi^k \in X$ we have that $\psi^0 \in n$ and $n \vdash \neg\psi$, or $\psi^1 \in n$ and $n \vdash \psi$.

At each step of the algorithm, if newly considered nodes do not satisfy the formula, then new candidate trees are built recursively by adding consistent parents to previously built trees. This process is achieved by the $Update$ function. In order to define this function, we first define auxiliary notions.

Given a formula ϕ and two nodes n_1 and n_2, we say that the nodes are *modally consistent* for $m \in \{\downarrow, \rightarrow\}$, written $\Delta_m(n_1, n_2)$, if and only if, for all $\langle m\rangle\psi$ and $\langle\overline{m}\rangle\psi$ in $lean(\phi)$, we have $\langle m\rangle\psi \in n_1 \Leftrightarrow n_2 \vdash \psi$, and $\langle\overline{m}\rangle\psi \in n_2 \Leftrightarrow n_1 \vdash \psi$.

Given a formula ϕ, and two nodes n_1 and n_2, we define the *counter consistency* of the nodes as follows: $\#_\downarrow(n_1, n_2)$ holds iff

- for all $\psi^{\langle k\rangle} \in lean(\phi)$, we have that $\psi^{\langle k\rangle} \in n_1$ iff $\psi^{k'} \in n_2$ and $k < k'$; and
- for all $\psi^{[k]} \in lean(\phi)$, we have that $\psi^{[k]} \in n_1$ iff $\top^{k_0}, \psi^{k_1}, (\neg\psi)^{k_2} \in n_2$, $k_0 = k_1 + k_2$, and $k_2 \leq k$.

$\#_\rightarrow(n_1, n_2)$ holds iff

- for all $\psi^k \in lean(\phi)$, we have that $\psi^k \in n_1$ and $n_1 \vdash \psi$ iff $\psi^{k-1} \in n_2$, or
- $\psi^k \in n_1$ and $n_1 \vdash \neg\psi$ iff $\psi^k \in n_2$.

The update function is in charge of checking that the newly formed triples are modally consistent, and that the counters are also consistent. Given the set X of triples and the set Y of nodes, we define $Update(X, Y)$ as the pair (X', Y'): X' is a set of triples (n, X_1, X_2) and Y' is the set containing the nodes in Y except n, such that $X_1, X_2 \in X$, $\Delta_\downarrow(n, n_1), \Delta_\rightarrow(n, n_2)$, and $\#_\downarrow(n, n_1), \#_\rightarrow(n, n_2)$.

5.3 Correctness and Complexity

Termination of the algorithm is straightforward since the set of nodes is finite and the update function is monotone. We now show that the algorithm is sound and complete.

Theorem 5.2 (Soundness). *If the algorithm returns that ϕ is satisfiable, then there is a Kripke tree structure satisfying ϕ.*

Proof sketch. Assume the algorithm builds a triple X such that $X \Vdash \phi$, then we build a Kripke tree structure $K = (\mathcal{N}, \mathcal{R}, L)$ from X, such that K satisfies ϕ.

$\mathcal{N}$ is the set of nodes of X. For every triple in (n, X_1, X_2) in X, we have that $\mathcal{R}(n, \downarrow) = n_1$ and $\mathcal{R}(n, \rightarrow) = n_2$, where n_i is the root of the non empty triples X_i $(i = 1, 2)$. For every $p \in lean(\phi)$, if $p \in n$, then $L(n) = p$.

We now show K satisfies ϕ by induction on the structure of ϕ. All cases are immediate by induction and by relying on the fact that fixpoints have an equivalent finite unfolding $\left(\mu x.\psi \equiv \psi\left[{}^{\mu x.\psi}/_{x}\right]\right)$. □

Theorem 5.3 (Completeness). *If a formula ϕ is satisfiable, then the algorithm returns that ϕ is satisfiable.*

Proof sketch. We first build a triple X from the smallest satisfying Kripke structure K, such that $X \Vdash \phi$. We then show the algorithm can build X.

X is defined homomorphic to K, such that its nodes contain the lean formulas satisfied by the corresponding nodes of K. More precisely, for $\mathcal{R}(n, \downarrow) = n_1$ and $\mathcal{R}(n, \rightarrow) = n_2$, there is a triple in X such that (n, X_1, X_2) and n_i is the root of X_i. For every node n in K, we have that $\psi \in n^\phi$ for every $\psi \in lean(\phi)$, such that n^ϕ in X is the corresponding node of n in K. We proceed to add the counters to each node: for every non empty sequence of sibling nodes $n_1, \ldots, n_k$ in K, such that n_{i+1} is the following sibling of n_i, and for every counter $\psi^k \in lean(\phi)$, if ψ is satisfied at k nodes of the sequence, then $\psi^k \in n_1^\phi$, where n_1^ϕ is the corresponding node of n_i.

It is now shown that X entails ϕ. We proceed by induction on the structure of ϕ. Most cases are immediate by the construction of X and by induction. For the fixpoint case $\mu x.\psi$, we test the equivalent formula $\psi\left[{}^{\mu x.\psi}/_{x}\right]$ and proceed again by structural induction. This is also immediate since variables, and then unfolded fixpoints, can only occur under the scope of a modality or a graded formula.

We now show the algorithm builds X by induction on the height of K. The base case is when K is a leaf, and it is immediate. For the induction step, first notice that the transition relation $\mathcal{R}$ is consistent with Δ_m. We know by induction that the left and right subtrees of X have been already produced by the algorithm. The root of X is still available in Y for the algorithm. This is because counters distinguish identical nodes, and we have a consistent bound for the counters (Theorem 5.1). We then conclude X is built by the algorithm. □

Theorem 5.4 (Complexity of satisfiability). *Satisfiability for the logic is EXPTIME-hard.*

Proof sketch. First recall the cardinality of the lean is at most polynomial w.r.t. to the formula size (Lemma 5.1). We then show the algorithm is exponential w.r.t. the lean size.

The cardinality of N^ϕ is exponential w.r.t. the lean size. Then the number of steps of the algorithm is bounded by 2^m, where m is the lean size. The cost of $Leaves(N^\phi)$ relies on a traversal on N^ϕ, and hence it is also exponential. Computing each relation $\Vdash$ and $\not\Vdash$ costs at most an exponential, since one traversal of the nodes of X is performed and the cost of computing the relation $\vdash$ is linear w.r.t. to the node size. The cost of computing the $Update$ function consists in: the searches required to form triples (n, X_1, X_2), and the functions Δ and $\#$. The searches are done on Y and X, each of them having exponential size. The cost of Δ and $\#$ relies on the one of the relation $\vdash$, which is linear. Hence, each step of the iteration costs at most an exponential time. Other set operations on X and Y can be done in linear time. Therefore, the total complexity is in EXPTIME. In addition, since the logic can encode all finite tree automata and is closed under negation, satisfiability for the logic is hard for EXPTIME. □

Corollary 5.1. *Reasoning (entailment, containment, equivalence) on regular path queries and regular tree types, with counting and interleaving, is decidable in EXPTIME.*

6 Related Work

Recent research has focused on expressing counting on children nodes only [Dal-Zilio *et al.*, 2004; Seidl *et al.*, 2004; Demri and Lugiez, 2010]. In these works, modal tree logics are equipped with Presburger arithmetic in order to express cardinality constraints. However, all these approaches do not support converse navigation and cannot fully support the navigation core of XPath. The approach presented here allows the multi-directional and recursive navigation required for the full navigation core of XPath. In addition, we also support counting constraints on children w.r.t. a constant.

The sheaves automata were introduced in [Dal-Zilio and Lugiez, 2003] to study the relationship of counting and interleaving in order to solve the entailment (satisfiability) problem of tree types. For that purpose, Presburger arithmetic formulas are used to model the interleave operator. However, the use of type constructs at top level of the interleave operator is not allowed. The logic introduced here has a less restrictive use of interleaving on regular tree types, since concatenation can be used at top level. [Colazzo *et al.*, 2009] identified fragments of tree types supporting interleaving with linear time complexity for containment. This came at the cost of drastic restrictions on the occurrence of identical labels. The approach presented here can support counting constructs for regular tree types in addition to interleaving.

Finally, from a general perspective, we recall that whenever adding a useful shorthand increasing the succinctness of a given logic, one must consider the impact to the complexity of the decision procedure of the original logic itself, but also, as importantly, consider the complexity of the translation of the added feature in terms of the original logic. The term "combined complexity" encompasses the two. We are notably interested in the combined complexity of deciding a μ-calculus for finite trees with backward modalities, nominals, and counting operators.

Similar logics but without a counting operator were considered in [Vardi, 1998] and [Calvanese *et al.*, 2009]. A μ-calculus with inverse plus counting operators is considered in [Barceló and Libkin, 2005]. However, a naïve translation of this logic in terms of μ-calculus with inverse without counting operators (such as the one of [Vardi, 1998]) would be exponential. The goal of the present paper is precisely to show how this blowup can be avoided. Furthermore, the best known complexity of the decision procedure for the logic in [Vardi, 1998] is $2^{\mathcal{O}(n^4 * \log(n))}$ [Grädel *et al.*, 2002] whereas our decision procedure operates in $2^{\mathcal{O}(n)}$ w.r.t. formula size n. In addition, for translating the interleave operator, we use nominals, which are not supported in [Vardi, 1998]. We recall that the μ-calculus of [Vardi, 1998] extended with counting, backward modalities, and nominals is undecidable (see [Bonatti *et al.*, 2008]). The reason why the combination of these features remains decidable in our case is that we restrict ourselves to finite trees.

7 Conclusion

We introduced a modal tree logic equipped with recursive and converse navigation, and counting operators on children nodes. We also presented a satisfiability-testing algorithm for the logic with exponential time complexity. The logic captures regular tree types and query languages with counting restrictions on children nodes. An extension of regular tree languages with the interleave operator whose operands are disjunction-free can also be described by the logic. The logic is closed under negation and can be used to decide typical reasoning problems on XML trees in exponential time: entailment, containment, and equivalence.

References

[Barceló and Libkin, 2005] Pablo Barceló and Leonid Libkin. Temporal logics over unranked trees. In *LICS*, pages 31–40. IEEE Computer Society, 2005.

[Bonatti *et al.*, 2008] P. Bonatti, C. Lutz, A. Murano, and M. Vardi. The complexity of enriched mu-calculi. *Logical Methods in Computer Science*, 4(3), 2008.

[Calvanese *et al.*, 2009] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. An automata-theoretic approach to regular XPath. In *DBPL*, pages 18–35, 2009.

[Calvanese *et al.*, 2010] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Node selection query languages for trees. In *AAAI*, 2010.

[Colazzo *et al.*, 2009] D. Colazzo, G. Ghelli, L. Pardini, and C. Sartiani. Linear inclusion for XML regular expression types. In *CIKM*, 2009.

[Dal-Zilio and Lugiez, 2003] S. Dal-Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In *RTA*, 2003.

[Dal-Zilio *et al.*, 2004] S. Dal-Zilio, D. Lugiez, and C. Meyssonnier. A logic you can count on. In *POPL*, 2004.

[Demri and Lugiez, 2010] S. Demri and D. Lugiez. Complexity of modal logics with Presburger constraints. *J. Applied Logic*, 8(3), 2010.

[Gelade, 2010] W. Gelade. Succinctness of regular expressions with interleaving, intersection and counting. *Theor. Comput. Sci.*, 411(31-33), 2010.

[Genevès *et al.*, 2007] P. Genevès, N. Layaïda, and A. Schmitt. Efficient static analysis of XML paths and types. In *PLDI*, 2007.

[Grädel *et al.*, 2002] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata logics, and infinite games: a guide to current research*. Springer-Verlag, 2002.

[Hosoya *et al.*, 2005] H. Hosoya, J. Vouillon, and B. Pierce. Regular expression types for XML. *ACM Trans. Program. Lang. Syst.*, 27(1), 2005.

[Seidl *et al.*, 2004] H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. Counting in trees for free. In *ICALP*, 2004.

[Vardi, 1998] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, pages 628–641, 1998.