A Counting Logic for Trees

Everardo Bárcenas

CONACYT - Universidad Veracruzana,
Mexico

iebarcenaspa@conacyt.mx

Abstract. It has been recently shown that the fully enriched μ-calculus, an expressive modal logic, is undecidable. In the current work, we prove that this result does not longer hold when considering finite tree models. This is achieved with the introduction of an extension of the fully enriched μ-calculus for trees with numerical constraints. Contrastively with graded modalities, which restrict the occurrence of immediate successor nodes only, the logic introduced in this paper can concisely express numerical constraints on any tree region, as for instance the ancestor or descendant nodes. In order to show that the logic is in EXPTIME, we also provide a corresponding satisfiability algorithm. By succinct reductions to the logic, we identify several decidable extensions of regular tree languages with counting and interleaving operators. It is also shown that XPath extensions with counting constructs on regular path queries can be concisely captured by the logic. Finally, we show that several XML reasoning problems (XPath queries with schemas), such as emptiness and containment, can be optimally solved with the satisfiability algorithm.

Keywords. Automated reasoning, modal logics, arithmetical constraints, formal languages, XPath.

1 Introduction

This work is devoted to the development and analysis of automated decision procedures for expressive tree logics. More precisely, we are interested in satisfiability algorithms for logics able to efficiently express counting constraints. We use these algorithms to model several reasoning problems in formal languages, such as the equivalence and containment of expressions.

Report on Ph.D. thesis. Thesis developed at the Institut National de Recherche en Informatique et en Automatique (INRIA) under the supervision of Vincent Quint and Nabil Layaïda.

Regular expressions are used to denote set of finite sequences of symbols (strings or words) from a finite alphabet. Consider for example a regular expression $(pq)^\star$ over the signature (alphabet) $\Sigma = \{p, q\}$. This expression is interpreted as the following set: $\{\epsilon, pq, pqpq, pqpqpq, \ldots, pqpqpqpqpqpq \ldots\}$.

It is well known that some formal languages easily described in English may require voluminous regular expressions. For instance, as pointed out in [15], the language L_{2a2b} of all strings over $\Sigma = \{a, b, c\}$ containing at least two occurrences of a and at least two occurrences of b requires a large expression:

$$
\begin{aligned}
&(a|b|c)^\star a(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star \mid \\
&(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star \mid \\
&(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star \mid \\
&(a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star a(a|b|c)^\star \mid \\
&(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star \mid \\
&(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star.
\end{aligned}
$$

If we add intersection $\cap$ to the operators for forming regular expressions, then the language L_{2a2b} can be expressed more concisely with the following expression:

$$
\begin{aligned}
&((a|b|c)^\star a(a|b|c)^\star a(a|b|c)^\star) \cap \\
&((a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star).
\end{aligned}
$$

In logical terms, intersection offers a dramatic reduction in expression size, which is crucial when the complexity of the decision procedure depends on the formula size. More precisely, it was shown in [14] that simple regular languages (without intersection) produce double-exponential larger expressions than regular formalisms equipped with intersection.

If we now consider a formalism equipped with the ability to describe numerical constraints on the frequency of occurrences, we get another exponential reduction in size [18]. For instance, the above expression can be formulated as follows:

$$((a|b|c)^\star a(a|b|c)^\star)^2 \cap ((a|b|c)^\star b(a|b|c)^\star)^2.$$

Although counting constraints does not increase the expressive power of regular languages, it has a drastic impact on succinctness, thus making reasoning over these languages harder. Indeed, reasoning on this kind of counting extensions without relying on their expansion (in order to avoid syntactic blow-ups) is often tricky. Determining satisfiability, containment, and equivalence over these classes of extended regular languages typically requires involved algorithms with higher complexity than ordinary regular languages. In [12], different extensions of regular expressions with intersection, counting constraints, and interleaving have been considered over strings, and for describing content models of sibling nodes in XML schema languages. The complexity of the inclusion problem over these different language extensions and their combinations typically ranges from polynomial time to exponential space.

Most XML schema languages, such as DTDs, XML Schema or RELAX NG, can be effectively embedded into regular tree languages (types) [22], which can be seen as regular expressions interpreted on sets of trees. In this work, we provide an optimal logic-based reasoning framework for regular tree languages equipped with counting operators.

The XPath language is the standard query language for XML documents, and it is also an important part of other XML technologies such as XSLT and XQuery. XPath expressions look like *directory navigation paths* over unranked tree structures. From a given context node, XPath expressions select subsets of tree nodes. One of the reasons why XPath is popular for web programming resides in its ability to express multi-directional navigation. Indeed, XPath expressions may use recursive navigation to access descendant nodes, and also backward navigation to reach previous siblings or ancestor nodes.

XPath expressions can also express cardinality constraints. In contrast with regular expressions and types, cardinality constraints in XPath expressions can be imposed in nodes other than the children ones. From [25], we know that expressing cardinality restrictions on nodes accessible by recursive multidirectional paths may introduce an extra-exponential cost or may even lead to undecidable formalisms. The logic-based reasoning framework introduced in the current work is also able to efficiently capture decidable extensions of XPath with counting constructs.

2 Related Work and Motivation

Here we review state of the art on the modern use of mathematical logic in automated reasoning on tree structures. In particular, we focus on reasoning frameworks able to express counting constraints. Moreover, we closely follow the application of these reasoning frameworks in the analysis of XML programming languages.

The encoding of Peano postulates about Arithmetic (PA) into First Order Logic (FOL) resulted in the first order theory of multiplication and addition, which is well-known to be undecidable: there is no algorithm able to decide whether an arithmetic expression is valid or not. It is also will-known that Presburger showed that the first order theory of addition, that is, PA without multiplication, is decidable. This theory was called Presburger Arithmetic (PresA). Since then, many other related decidable theories have been identified. For instance, it was shown in [19] that the existential theory of addition and divisibility is also decidable. PresA found application in practically all areas of Computer Science. For instance, in Program Verification, balanced tree structures, in contrast with their unbalanced counterpart, require frameworks able to capture counting constraints [20]. In the XML setting, extensions of tree logics with Presburger constraints were reported in [8] and [24].

The main motivation for study of second order theories to model formal languages came with the classical result of Büchi that established the exact correspondence between regular languages (those definable by tree automata) and Monadic Second Order Logic (MSO). Seidl in [23] designed a logic

combining MSO for finite trees with Presburger Arithmetic, where the cardinality constraints are imposed only on children nodes. This logic was introduced to support reasoning on XML documents, that is, unranked trees. Unfortunately, this logic turned out to be undecidable.

Modal logics have been identified as fragments of FOL and MSOL with nice computational properties, which places them as more attractive alternatives. The μ-calculus is a propositional multimodal logic extended with least μ and greatest fixpoints ν. These fixpoints provide a great expressive power: actually, μ-calculus turned out to encompass all program-based logics, that is, all families of linear temporal logics, Computational Tree Logic (CTL), Propositional Dynamic Logic (PDL), and many Description Logics (DL). The expressive power of the μ-caluclus coincide with MSO [17], whereas its complexity is much better, EXPTIME-complete [6]. The fully enriched μ-calculus is an extension consisting of nominals, inverse and graded modalities. Nominals are used to denote individuals, backward navigation is achieved through inverse modalities, and graded modalities are specialized constructs to denote numerical constraints on immediate neighbor nodes. However, this expressive logic is undecidable [6]. In the current work, we show this result does not hold any more when considering tree models, that is, we showed that the fully enriched μ-calculus for trees is decidable, moreover, we show it is in EXPTIME.

In the context of trees, graded modalities are limited to express constraints on children nodes only. Constraints on further nodes have been recently studied in [10, 5]. In these works CTL is extended with numerical constraints on downward CTL expressions, for instance, on expressions denoting descendant nodes. In this work we also propose an extension of the fully enriched μ-calculus with global numerical constraints. These constraints can denote numerical restrictions in any tree region, for instance, in descendants or ancestors. In this work we show that the fully enriched μ-calculus for trees with global numerical constraints is also in EXPTIME.

More general constraints in the context of tree models have been recently proposed in [8, 24, 9, 3]. These constraints are written in terms of Presburger arithmetical expressions, however, they are limited to constrain children nodes only.

3 The μ-calculus with Global Numerical Constraints

In this section we introduce an extension of the fully enriched μ-calculus for finite unranked tree models with global numerical constraints. That is, this logic is equipped with operators for recursion, inverse programs, global counting constraints, and nominals.

Definition 3.1 (Syntax)**.** The set of formulas in the fully enriched μ calculus is defined by the following grammar:

$$\phi := p \mid x \mid \neg\phi \mid \phi \vee \phi \mid \langle m\rangle\phi \mid \mu x.\phi \mid \phi > k.$$

Variables are assumed to be bounded and under the scope of a modal ($\langle m\rangle\phi$) or a counting ($\phi > k$) formula. Formulas are interpreted as subset tree nodes: propositions p are used to label nodes; negation and disjunction are interpreted as complement and union of sets, respectively; modal formulas $\langle m\rangle\phi$ are true in nodes, such that ϕ holds in at least one accessible node through adjacency m, which may be either $\downarrow$, $\rightarrow$, $\uparrow$ or $\leftarrow$, which in turn are interpreted as the children, right sibling, parent and left sibling relations, respectively; $\mu x.\phi$ is interpreted as a least fixed-point; and counting formulas $\phi > k$ hold in every node of the tree model if and only if ϕ holds in *at least* $k+1$ nodes in the *entire tree* (see Definition 3.2).

We also use the following notation: $\top = p \vee \neg p$, $\phi \wedge \psi = \neg(\neg\phi \vee \neg\psi)$, $[m]\,\phi = \neg\langle m\rangle\neg\phi$, $\nu x.\phi = \neg\mu x.\neg\phi\,[^{x}/_{\neg x}]$, $(\phi \leq k) = \neg(\phi > k)$, and $\phi \# k$ for $\# \in \{\leq, >\}$. Note that $\top$ is true in every node, conjunction $\phi \wedge \psi$ holds whenever both ϕ and ψ are true, $[m]\phi$ holds in nodes where ϕ is true in each accessible node through m, $\nu x.\phi$ is a greatest fixed-point, and $\phi \leq k$ holds in nodes where the number of its ϕ children nodes is less or equal than the constant k.

We can express existential statements with counting formulas. For instance, if we want to

select the nodes expressed by a formula ψ, only if *there is a node* satisfying ϕ, then we write

$$(\phi{>}\,0) \wedge \psi.$$

Universality can also be expressed. The following formula selects the ψ nodes when *every node* satisfies ϕ:

$$[(\neg\phi){\leq}\,0] \wedge \psi.$$

Note that with counting formulas it is also possible to restrict the number of nodes occurring in a particular region. First, consider, for instance, the *descendants region*. This can be expressed as follows:

$$\mu x.\langle\uparrow\rangle(p_0 \vee x).$$

This formula denotes *the descendants of the* p_0 *nodes*. Recall that $\uparrow$ denotes the parent relation. Hence, the formula holds in nodes from where, by recursive navigations through parents, nodes named p_0 are accessible. Then, if we want to restrict the number of descendants of the p_0 nodes in a tree, we write

$$[\mu x.\langle\uparrow\rangle(p_0 \vee x)]{\leq}\,6.$$

Now, if we want to restrict the number of some descendants, say, descendants named p_1, then we write

$$([\mu x.\langle\uparrow\rangle(p_0 \vee x)] \wedge p_1){\leq}\,6.$$

Note that $([\mu x.\langle\uparrow\rangle(p_0 \vee x)] \wedge p_1){\leq}\,6$ holds due to *all* p_1 descendants of *each* p_0 node. That is, if in a model there are 2 nodes named p_0 with 2 and 4 descendants named p_1, respectively, then the formula $([\mu x.\langle\uparrow\rangle(p_0 \vee x)] \wedge p_1){\leq}\,6$ holds due to all 6 descendants of both p_0 nodes (see Fig. 1).

However, one may also want to restrict the number of descendants of a particular node. This can be done by isolating the origin node from where navigation starts (during counting). For this purpose we first define the following formula:

$$(o{\leq}\,1) \wedge (o{>}\,0).$$

In this formula, proposition o occurs exactly once in a model. If we want to identify where o occurs, then we write:

$$(o{=}\,1) \wedge o,$$

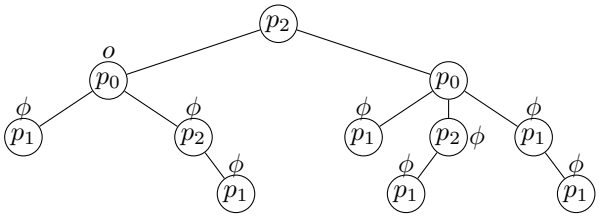

Fig. 1. Tree model example: descendant region of p_0 nodes is denoted by the formula $\phi \equiv \mu x.\langle\uparrow\rangle(p_0 \vee x)$; formula $(\phi \wedge p_1){\leq}\,6$ holds because the p_0 nodes have exactly 6 descendants labeled with p_1; $(\mu x.[(p_0 \wedge o \wedge o = 1) \vee \langle\uparrow\rangle x] \wedge p_1){\leq}\,2$ is true because there is a p_0 node, the one marked with o, with 2 descendants named p_1

where $o{=}\,1$ stands for $(o{\leq}\,1) \wedge (o{>}\,0)$. Note that formula $(o{=}\,1){\wedge}o$ selects a node only if the formula is true in exactly that node, then this formula can be seen as a nominal [6].

Now since we can isolate a single node in a model, we can thus restrict the counting from a particular node, consider, for instance, the following formula:

$$[\mu x.\langle\uparrow\rangle([(o{=}\,1) \wedge o \wedge p_0] \vee x)]{\leq}\,2.$$

This formula is true in models where there is a single p_0 node with no more than 2 descendants. If in addition, we want to name the descendants, say, p_1, then we write

$$[\mu x.\langle\uparrow\rangle([(o{=}\,1) \wedge o \wedge p_0] \vee x) \wedge p_1]{\leq}\,2.$$

A graphical representation of the examples above is depicted in Fig. 1.

In order to provide a formal semantics, we need some preliminaries. A tree structure $\mathcal{T}$ is a tuple $(P, \mathcal{N}, \mathcal{R}, L)$, where P is a set of propositions; $\mathcal{N}$ is a finite set of nodes; $\mathcal{R} : \mathcal{N} \times M \times \mathcal{N}$ is a relation between nodes and modalities $M = \{\downarrow, \rightarrow, \uparrow, \leftarrow\}$ forming a tree, written $n \in \mathcal{R}(n, m)$; and $L : \mathcal{N} \times P$ is a labeling relation, written $p \in L(n)$.

Given a tree structure, a valuation V of variables is defined as a mapping from the set of variables X to the nodes $V : X \mapsto \mathcal{N}$.

Definition 3.2 (Semantics)**.** Given a tree structure $\mathcal{T}$ and a valuation V, formulas in the fully enriched μ-calculus are interpreted as follows:

$$\begin{aligned}
&[\![p]\!]^{\mathcal{T}}_V = \{n \mid p \in L(n)\},\\
&[\![x]\!]^{\mathcal{T}}_V = V(x),\\
&[\![\neg\phi]\!]^{\mathcal{T}}_V = \mathcal{N} \setminus [\![\phi]\!]^{\mathcal{T}}_V,\\
&[\![\phi \vee \psi]\!]^{\mathcal{T}}_V = [\![\phi]\!]^{\mathcal{T}}_V \cup [\![\psi]\!]^{\mathcal{T}}_V,\\
&[\![\langle m\rangle\phi]\!]^{\mathcal{T}}_V = \left\{n \mid \mathcal{R}(n,m) \cap [\![\phi]\!]^{\mathcal{T}}_V \neq \emptyset\right\},\\
&[\![\mu x.\phi]\!]^{\mathcal{T}}_V = \bigcap \left\{\mathcal{N}' \subseteq \mathcal{N} \,\middle|\, [\![\phi]\!]^{\mathcal{T}}_{V[\mathcal{N}'/x]} \subseteq \mathcal{N}'\right\},\\
&[\![\phi > k]\!]^{\mathcal{T}}_V = \begin{cases} \mathcal{N} & \text{if } |[\![\phi]\!]^{\mathcal{T}}_V| > k,\\ \emptyset & \text{otherwise.}\end{cases}
\end{aligned}$$

Recall that the fully enriched μ-calculus in the context of trees can express numerical constraints on children nodes only. It is clear that the logic introduced in the current work can also express these kind of constraints, consider, for instance, the following formula:

$$p_1 \wedge o \wedge o = 1 \wedge [p_2 \wedge \mu x.\langle\uparrow\rangle o \vee \langle\leftarrow\rangle x] > k.$$

This formula holds in a p_1 node with at least k children named p_2. Moreover, it has been recently shown that global numerical constraints are exponentially more succinct than the graded modalities [4].

4 Regular Path Queries with Counting

The navigation core of the XPath query language (for XML documents) has been formalized as regular path queries, and it is known to correspond to FOL^2 [25, 21]. In this Section, we introduce an extension of regular path queries with counting constructs.

We now describe the extension of regular paths with counting constructs.

Definition 4.1 (Syntax)**.** The syntax of regular path queries with counting on children paths ρ is given as follows:

$$\begin{aligned}
\alpha &:= \downarrow|\rightarrow|\uparrow|\leftarrow|\downarrow^\star|\uparrow^\star,\\
\varrho &:= \top \mid \alpha \mid p \mid \alpha : p \mid \varrho/\varrho \mid \varrho[\beta],\\
\beta &:= \varrho > k \mid \beta \vee \beta \mid \neg\beta,\\
\rho &:= \varrho \mid /\rho \mid \rho \cup \rho \mid \rho \cap \rho \mid \rho \setminus \rho,
\end{aligned}$$

where p is a proposition, and k is a positive integer in binary.

We also consider the following syntactic sugar: $\varrho \leq k$ is written instead of $\neg(\varrho > k)$; ϱ instead of $\varrho > 0$; $\beta_1 \wedge \beta_2$ instead of $\neg(\neg\beta_1 \vee \neg\beta_2)$; and $\varrho[\beta_1][\beta_2]$ instead of $\varrho[\beta_1 \wedge \beta_2]$.

Regular path expressions are interpreted as node-selection queries on tree structures. In particular, the axis relations α are interpreted as follows: children $\downarrow$, following sibling $\rightarrow$, parent $\uparrow$, previous sibling $\leftarrow$, descendants $\downarrow^\star$, and ancestors $\uparrow^\star$. Step paths $\alpha : p$ selects the p nodes reachable by α. Symbol $/$ is used to compose paths. A qualified path $\varrho[\beta]$ selects the nodes denoted by ϱ that satisfies the boolean condition β. A qualified path $[\varrho > k]$ is true when ϱ selects at least k nodes. A Boolean combination of qualifiers β is interpreted in the obvious manner. The path $/\rho$ selects the nodes denoted by ρ that are reachable from the root. Union, intersection, and difference of paths are interpreted as expected. Consider, for instance, the following composition of paths:

$$\uparrow^\star : p_1 / \downarrow^\star : p_2.$$

This query, evaluated from some context (a node subset), navigates to the p_1 ancestors of the context, and from there, it selects the p_2 descendants. Now consider the following qualified path:

$$\uparrow^\star : p_1[\downarrow^\star : p_2].$$

In constrast with the previous example, this query selects the p_1 ancestors *with at least* 1 *descendant named* p_2.

Before giving a formal description of the semantics of queries, we introduce the following notation: in a Kripke structure, $n_1 \xrightarrow{\alpha} n_2$ means than n_1 is related by means of α with n_2, where α can be any axis relation ($\downarrow, \rightarrow, \uparrow, \leftarrow, \downarrow^\star, \uparrow^\star$).

Definition 4.2 (Semantics)**.** The semantics of regular path queries is defined by a function $[\![\cdot]\!]^{\cdot}$ from

the queries with respect to a tree $\mathcal{T}$, to pairs of nodes in $\mathcal{T}$.

$$
\begin{aligned}
&[\![\top]\!]^{\mathcal{T}} = \mathcal{N} \times \mathcal{N},\\
&[\![p]\!]^{\mathcal{T}} = \{(n,n) \mid p \in L(n)\},\\
&[\![\alpha]\!]^{\mathcal{T}} = \{(n_1,n_2) \mid n_1 \xrightarrow{\alpha} n_2\},\\
&[\![\alpha : p]\!]^{\mathcal{T}} = \{(n_1,n_2) \in [\![\alpha]\!]^{\mathcal{T}} \mid p \in L(n_2)\},\\
&[\![\varrho_1/\varrho_2]\!]^{\mathcal{T}} = [\![\varrho_1]\!]^{\mathcal{T}} \circ [\![\varrho_2]\!]^{\mathcal{T}},\\
&[\![\varrho[\beta]]\!]^{\mathcal{T}} = \{(n_1,n_2) \in [\![\varrho]\!]^{\mathcal{T}} \mid n_2 \in [\![\beta]\!]^{\mathcal{T}}\},\\
&[\![\varrho]\!]^{\mathcal{T}} = \{n_1 \mid (n_1,n_2) \in [\![\varrho]\!]^{\mathcal{T}}\},\\
&[\![\varrho > k]\!]^{\mathcal{T}} = \{n_1 \mid |\{n_2 \mid (n_1,n_2) \in [\![\varrho]\!]^{\mathcal{T}}\}| > k\},\\
&[\![\neg\beta]\!]^{\mathcal{T}} = \mathcal{N} \setminus [\![\beta]\!]^{\mathcal{T}},\\
&[\![\beta_1 \vee \beta_2]\!]^{\mathcal{T}} = [\![\beta_1]\!]^{\mathcal{T}} \cup [\![\beta_2]\!]^{\mathcal{T}},\\
&[\![/\varrho]\!]^{\mathcal{T}} = \{(r,n) \in [\![\varrho]\!]^{\mathcal{T}} \mid r \text{ is the root}\},\\
&[\![\rho_1 \cup \rho_2]\!]^{\mathcal{T}} = [\![\rho_1]\!]^{\mathcal{T}} \cup [\![\rho_2]\!]^{\mathcal{T}},\\
&[\![\rho_1 \cap \rho_2]\!]^{\mathcal{T}} = [\![\rho_1]\!]^{\mathcal{T}} \cap [\![\rho_2]\!]^{\mathcal{T}},\\
&[\![\rho_1 \setminus \rho_2]\!]^{\mathcal{T}} = [\![\rho_1]\!]^{\mathcal{T}} \setminus [\![\rho_2]\!]^{\mathcal{T}}.
\end{aligned}
$$

Notice that the function $[\![\cdot]\!]^{\cdot}$ is introduced to distinguish the interpretation of paths inside qualifiers.

Definition 4.3 (Reasoning problems)**.** We now define the emptiness, containment, and equivalence problems of regular path queries as follows.

- We say a query ρ is empty if and only if for every tree $\mathcal{T}$, its interpretation is empty, that is, $[\![\rho]\!]^{\mathcal{T}} = \emptyset$;
- It is said that a query ρ_1 is contained in a query ρ_2 if and only if for every tree $\mathcal{T}$, each pair of nodes in the interpretation of ρ_1 is in the interpretation of ρ_2, that is, $[\![\rho_1]\!]^{\mathcal{T}} \subseteq [\![\rho_2]\!]^{\mathcal{T}}$; and
- Two queries ρ_1 and ρ_2 are equivalent if and only if for every tree $\mathcal{T}$, ρ_1 is contained in ρ_2 and the other way around, that is, $[\![\rho_1]\!]^{\mathcal{T}} \subseteq [\![\rho_2]\!]^{\mathcal{T}}$ and $[\![\rho_2]\!]^{\mathcal{T}} \subseteq [\![\rho_1]\!]^{\mathcal{T}}$.

Regular path queries (without counting) can be written in terms of the μ-calculus [2, 1]. For instance, the query $\downarrow^{\star}\!: p$, evaluated in the root r, selects the p descendants of r. This can be written as follows:

$$[\mu x.\langle\uparrow\rangle(r \vee x)] \wedge p.$$

If we want to evaluate the query in another context (node subset), represented by a C formula, then we simply replace the occurrence of r by C. For instance, let us say the context is represented by all the nodes named p_0, then the p ancestors of p_0 nodes can be written as follows:

$$[\mu x.\langle\uparrow\rangle(p_0 \vee x)] \wedge p.$$

We now show that counting constructs can also be captured by the fully enriched μ-calculus.

Definition 4.4 (Logical embedding of queries)**.** Given a context formula C, the translation F from regular path queries into the logic is defined as follows:

$$
\begin{aligned}
F(\downarrow, C) =& \langle\uparrow\rangle C,\\
F(\rightarrow, C) =& \langle\leftarrow\rangle C,\\
F(\uparrow, C) =& \langle\downarrow\rangle C,\\
F(\leftarrow, C) =& \langle\rightarrow\rangle C,\\
F(\downarrow^{\star}, C) =& \mu x.\langle\uparrow\rangle(C \vee x),\\
F(\uparrow^{\star}, C) =& \mu x.\langle\downarrow\rangle(C \vee x),\\
F(\alpha : p, C) =& F(\alpha, C) \wedge p,\\
F(\varrho_1/\varrho_2, C) =& F(\varrho_2, F(\varrho_1, C)),\\
F(\varrho[\beta], C) =& F(\varrho, C) \wedge o \wedge F(\beta, [o{=}1] \wedge o),\\
F(\varrho > k, C) =& F(\varrho, C) > k,\\
F(\neg\beta, C) =& F'(\beta, C),\\
F(\beta_1 \vee \beta_2, C) =& F(\beta_1, C) \vee F(\beta_2, C),\\
F(/\varrho, C) =& F(\varrho, C \wedge \neg(\langle\uparrow\rangle\top \wedge \langle\leftarrow\rangle\top)),\\
F(\rho_1 \cap \rho_2, C) =& F(\rho_1, C) \wedge F(\rho_2, C),\\
F(\rho_1 \cup \rho_2, C) =& F(\rho_1, C) \vee F(\rho_2, C),\\
F(\rho_1 \setminus \rho_2, C) =& F(\rho_1, C) \wedge F'(\rho_2, C),
\end{aligned}
$$

where

$$
F'(\rho) = \begin{cases} F'(\varrho, C \wedge \neg[\langle\uparrow\rangle\top \wedge \langle\leftarrow\rangle\top]) \\ \quad \text{if } \rho \text{ has the form } /\varrho, \\ \neg F(\rho) \quad \text{otherwise.} \end{cases}
$$

$$
F'(\varrho) = \begin{cases} \neg F(\varrho', C) \vee [o \wedge \neg F(\beta, [o{=}1] \wedge o)] \\ \quad \text{if } \varrho \text{ has the form } \varrho'[\beta], \\ \neg F(\varrho) \quad \text{otherwise.} \end{cases}
$$

Intuitively, F' represents the negation of F, however, in the case where there is a counting operator, the fresh proposition o, which serves to fix an

origin node, is not negated. Note that the constraint $o = 1 \wedge o$ is not affected by negation because it always occurs in the scope of a counting operator.

Consider the following query evaluated in a context C:

$$\downarrow\colon p_1[\downarrow^\star\colon p_2 > k].$$

The query selects the p_1 children of C with at least $k+1$ descendants named p_2. The first part of the query $\downarrow\colon p_1$ is translated as follows:

$$p_1 \wedge \langle\uparrow\rangle C,$$

that is, the p_1 nodes with C as parent.

The translation of the counting expression $\downarrow^\star\colon p_2 > k$ is

$$o \wedge [p_2 \wedge \mu x.\langle\uparrow\rangle([o{=}1 \wedge o] \vee x)] > k.$$

This formula holds if and only if there are more than k descendant nodes, named p_2, of a single node named o. Then, the translation of the entire query is the following:

$$F(\downarrow\colon p_1[\downarrow^\star\colon p_2 > k]) = (p_1 \wedge \langle\uparrow\rangle C) \wedge o \wedge [p_2 \wedge \mu x.\langle\uparrow\rangle([o{=}1 \wedge o] \vee x)] > k.$$

The proposition o is used to fix a context for the counting subformula. o holds in a single p_1 node, then the p_2 descendants of that particular p_1 node are the only ones counted.

With the translation function F, we can now use the logic as a reasoning framework to solve emptiness, containment, and equivalence of regular path queries with counting constructs on children paths; moreover, since translation F does not introduce duplications, it is easy to see that the formula resulting from the translation has a linear size with respect to the input query.

Theorem 4.1 (Query reasoning)**.** *For any regular path queries ρ, ρ_1, ρ_2, tree $\mathcal{T}$, and valuation V, the following holds:*

- $[\![\rho]\!]^{\mathcal{T}} = \emptyset$ *if and only if* $[\![F(\rho, \top)]\!]^{\mathcal{T}}_V = \emptyset$*;*
- $[\![\rho_1]\!]^{\mathcal{T}} \subseteq [\![\rho_2]\!]^{\mathcal{T}}$ *if and only if* $[\![F(\rho_1, \top) \wedge F'(\rho_2, \top)]\!]^{\mathcal{T}}_V = \emptyset$*; and*
- $F(\rho, \top)$ *has linear size with respect to* ρ *and* $F'(\rho_1, \top) \wedge F(\rho_2, \top)$ *has linear size with respect to* ρ_1 *and* ρ_2*.*

Proof. For the first item, we proceed by structural induction on ρ.

In order to prove the case when ρ has the form ϱ, we will prove the following: ϱ evaluated in a context C is satisfiable by a tree T if and only if $F(\varrho, C)$ is satisfiable by T.

Consider ρ is the basic query $\downarrow^\star\colon p$, then $F(\downarrow^\star\colon p, C) = p \wedge \mu x.\langle\uparrow\rangle(C \vee x)$, which clearly selects exactly the same nodes as ρ evaluated in C. The proof for the cases with the other axes ($\downarrow, \uparrow, \rightarrow, \leftarrow, \uparrow^\star$) is similar.

Now let the input query be a composition of paths, that is, ρ has the form ϱ_1/ϱ_2. Intuitively, ϱ_1/ϱ_2 selects the nodes denoted by ρ_2 evaluated from the nodes satisfying ϱ_1, that is, ϱ_1 is the context. That is precisely what it means by $F(\varrho_2, F(\varrho_1, C))$. By induction $F(\varrho_1, C)$ corresponds to ϱ_1, and then also by induction $F(\varrho_2, F(\varrho_1, C))$ corresponds to ϱ_1/ϱ_2 evaluated in C.

Before proving the case when the input query has the form $\varrho_1[\varrho_2 > k]$, we need first to prove that $\varrho_2 > k$ is satisfiable by T if and only if $F(\varrho_2, o = 1 \wedge o) > k$ is satisfiable by T. This is achieved by induction on the structure of ϱ_2. Consider ϱ_2 has the form $\downarrow\colon p$. Then $F(\downarrow\colon p, \top) = p \wedge \langle\uparrow\rangle\top$. This formula selects all the p children of the model. However, according to the semantics of CPath queries (Definition 4.2), we need to count the p children of a single node. This is achieved by fixing the context with a new fresh proposition o occurring only once in the model $o = 1$. Hence $[p \wedge \langle\uparrow\rangle([o = 1] \wedge o)] > k$ is satisfiable by T if and only if $\downarrow\colon p > k$ is satisfiable by T. We proceed analogously for the other axes. For the other cases of ϱ_2, that is, when ϱ_2 is a composition of paths (ϱ_2'/ϱ_2'') and a qualified path ($\varrho_2'[\beta']$), the proof goes straightforward by induction.

Now that we know that $\varrho_2 > k$ is satisfiable by T if and only if $F(\varrho_2) > k$ is satisfiable by T, and that by induction, ϱ_1 evaluated in C is satisfiable by T if and only if $F(\varrho_1, C)$ is satisfiable T, we can thus infer that $F(\varrho_1, C) \wedge o \wedge F(\varrho_2, [o = 1] \wedge o)$ is satisfiable by T if and only if $\varrho_1[\varrho_2 > k]$ is satisfiable in context

C by T. Note that o is used to select a single ϱ_1 node.

When ϱ has the form $\varrho_1[\beta]$, the cases, when β is a disjunction or a negation, are immediate by induction. In the case of negation, it is important to notice that the negation of $F(\varrho', [o = 1] \wedge o) > k$ does not affect the context, that is, the negation never goes inside the formula $[o = 1] \wedge o$.

Consider now the case when the input query has the form $\rho_1 \setminus \rho_2$. The only interesting case is when ρ_2 has the form $\varrho_1[\varrho_2 > k]$. It is easy to see, by induction, that $F(\rho_1, C)$ is satisfiable by T if and only if ρ_1 is satisfiable by T. Also by induction we know that $\neg F(\varrho_1, C) \vee (o \wedge \neg F(\varrho_2, [o = 1] \wedge o))$ is satisfiable by T if and only if $\varrho_1[\varrho_2 > k]$ is not satisfiable by T. We can hence conclude that $F(\rho_1, C) \wedge [\neg F(\varrho_1, C) \vee (o \wedge \neg F(\varrho_2[o = 1] \wedge o))]$ is satisfiable by T if and only if $\rho_1 \setminus (\varrho_1[\varrho_2 > k])$ is also satisfiable.

The cases, when the input query has the forms $\rho_1 \cup \rho_2$, $\rho_1 \cap \rho_2$, and $/\rho_1$, are straightforward by induction.

For the second item, we proceed analogously as for the first item in the case when the input query has the form $\rho_1 \setminus \rho_2$.

The third item is proven immediately by structural induction on the input query and by noticing that function F does not introduce duplications. □

5 Regular Tree Languages with Counting

Analogously as regular expressions denote sets of strings, regular tree expressions denote sets of trees. In the context of XML, regular tree languages are used as schema document languages, such as DTDs, XML schema and RELAX NG. Regarding numerical restrictions on children nodes, XML schema provides explicit notations: MinOccur and MaxOccur. Although these kind of numerical restrictions does not provide more expressive power, they have been recently shown to be exponentially more succinct [12], that is, reasoning on regular tree expressions with counting constructs is exponentially easier. Other interesting extension of regular languages is the interleaving operator, which, as the name suggests, denotes all possible interleaves of the corresponding subexpressions, for instance, $pq \,\&\, rs$ denotes the following expressions: $pqrs$, $prqs$, $prsq$, $rpqs$, $rpsq$, or $rspq$. The interleaving operator clearly does not introduce extra expressive power to regular tree languages, however, it is also clear that this operator does introduce an improvement regarding succinctness. Actually, interleaving is exponentially more succinct [12].

In this section, we introduce explicit constructs for numerical restrictions and interleaving on regular tree expressions. We then show this extension of regular tree languages can be succinctly expressed in terms of the fully enriched μ-calculus for trees extended with global numerical constraints, which can thus be used as an optimal reasoning framework.

Definition 5.1 (Syntax)**.** Regular tree expressions with counting and interleaving constructs are defined by the following grammar:

$$\begin{aligned}\mathcal{T} \quad := \quad & \epsilon \mid x \mid p[\mathcal{T}^{\#k}] \mid \mathcal{T}_1 \cdot \mathcal{T}_2 \mid \mathcal{T}_1 + \mathcal{T}_2 \\ & \mid p[\mathcal{T}_1' \& \mathcal{T}_2' \& \ldots \& \mathcal{T}_n'] \mid \text{let } \overline{x}.\overline{\mathcal{T}} \text{ in } \mathcal{T},\end{aligned}$$

where ϵ denotes the empty tree, p are propositions, $+$ is used for disjunction (alternation), concatenation is expressed as usual by $\cdot$, $\#$ stands for either $>$ or $\leq$, k is a natural number, and $\mathcal{T}_i'$ are disjunction-free tree expressions, which are defined as follows:

$$\mathcal{T}' := x \mid p[\mathcal{T}^{\#k}] \mid \mathcal{T}_1' \cdot \mathcal{T}_2' \mid p[\mathcal{T}_1' \& \mathcal{T}_2' \& \ldots \& \mathcal{T}_n'].$$

For example, the counting expression $p[q^{>5}]$ denotes the trees rooted at p with at least 5 children named q, whereas the interleaving expression $p[q \& r]$ stands for the trees rooted at p whose children can be q and r, or r and q, in that order. We can also define the following common syntactic sugar: $p[\epsilon] = p$, $\mathcal{T}^? = \epsilon + \mathcal{T}$, $\mathcal{T}^\star =$ let $x.\mathcal{T}$ in $\mathcal{T} \cdot (x + \epsilon)$, and $\mathcal{T}^+ = \mathcal{T} \cdot \mathcal{T}^\star$.

For a formal denotational semantics of regular tree expressions we refer the reader to the following works: [16, 1, 2, 4].

As first noted in [13], regular tree expressions can be succinctly captured by the μ-calculus for trees. We now show that the counting and interleaving operators can be linearly translated in terms of the fully enriched μ-calculus for trees.

Definition 5.2 (Logical embedding of regular tree expressions)**.** Given a linear translation F of regular tree expressions in μ-calculus formulae, the translation is extended for counting and interleaving operators as follows:

$$F\left(p[\mathcal{T}^{\#k}]\right) = p \wedge o \wedge [F(\mathcal{T}) \wedge \langle\uparrow\rangle(o = 1 \wedge o) \wedge \neg\langle\rightarrow\rangle\top]\#k,$$

$$F\left(p[\mathcal{T}_1 \& \ldots \& \mathcal{T}_n]\right) = p \wedge o \wedge (\langle\uparrow\rangle o = k) \bigwedge_{i=1}^{n} \left(F\left(\mathcal{T}_i\right) \wedge o_i \wedge \langle\uparrow\rangle o\right) = 1.$$

where k is the number of children induced by T_i and o is a fresh proposition used to distinguish identical labels.

For instance, the following expression $p[q\&r]$ is translated to $p \wedge o \wedge o = 1 \wedge (\phi \wedge o_1 \wedge \langle\uparrow\rangle o) = 1 \wedge (\psi \wedge o_2 \wedge o) = 1 \wedge (\langle\uparrow\rangle o = 2)$, where ϕ is $q \wedge \neg\langle\downarrow\rangle\top$ and ψ is $r \wedge \neg\langle\downarrow\rangle\top$. Formula $(\phi \wedge o_1 \wedge \langle\uparrow\rangle o) = 1$ then restricts the occurrence of q to exactly once, it also says that q is the children of o and hence of p. Nominal o_i is introduced to distinguish potential multiple occurrence of p. The same restrictions are set for q in the formula involving ψ. The number of children is restricted to 2 (induced by p and q) with formula $\langle\uparrow\rangle o = 2$. Hence, p and q occur exactly once, at any order (interleaved), and no other proposition occurs because there are only 2 children.

Reasoning problems for regular tree expressions are defined analogously as the reasoning problems for queries introduced in Section 4. Moreover, reasoning problems of regular path queries in the presence of regular tree expressions are particularly useful in many static analysis problems in XML. Since the translation function F of regular tree expressions does not introduce duplication, we can then conclude that the fully enriched μ-calculus can also be used as an XML reasoning framework for the extension of queries and schemas with counting and interleaving constructs, respectively.

Theorem 5.1 (XML reasoning)**.** *Emptiness, containment, and equivalence of regular path queries with counting constructs (on children paths) in the presence of regular tree expressions with counting and interleaving operators can be linearly reduced to satisfiability problems for the fully enriched μ-calculus for trees with global numerical constraints.*

Proof. This is an immediate consequence of Theorem 4.1, and by the fact that regular tree expressions with counting and interleaving can be linearly characterized by the logic. We now show this last fact by structural induction on the input tree expression. Regular tree expressions without counting and interleaving can be linearly expressed by μ-calculus formulas (without global counting). This has been previously proven in [2]. We now show that counting operators in regular tree expressions can be expressed by global counting. By induction we know $F(\mathcal{T})$ is satisfiable by a tree T if and only if e is satisfiable. Then the formula $[F(\mathcal{T}) \wedge \langle\uparrow\rangle(o = 1 \wedge o)]\#k$ is satisfiable by T if and only if there is a node with children matching $F(\mathcal{T})$ and satisfying the numerical constraint $\#k$. Therefore $p \wedge o \wedge [F(\mathcal{T}) \wedge \langle\uparrow\rangle(o = 1 \wedge o)]\#k$ is satisfiable by T if and only if $p[\mathcal{T}^{\#k}]$ is satisfiable by T. The case for interleaving is analogous. □

6 Satisfiability

In this section, we introduce a satisfiability algorithm for the μ-calculus with global numerical constraints for trees. First, we give a syntactical version of tree models, called Fischer-Ladner trees. Then we show how the algorithm can construct such Fischer-Ladner trees.

Since there is a well-known bijection between binary and n-ary unranked trees [1, 2] (depicted in Fig. 2), without loss of generality, we then define the algorithm for binary trees only, where the modality $\downarrow$ is now interpreted as the first child relation, $\rightarrow$ is now the right sibling relation, whereas their inverses $\uparrow$ and $\leftarrow$ are interpreted as expected.

For the algorithm, we consider formulas in the negation normal form (NNF) only. In the negation normal form $\text{nnf}(\phi)$ of a formula ϕ, negation occurs only immediately above propositions, $\top$ and modal subformulas $\langle m\rangle\top$. This is obtained by the following rules together with the usual DeMorgan's: $\neg\langle m\rangle\phi = \langle m\rangle\neg\phi \vee \neg\langle m\rangle\top$, $\neg(\phi > k) = \phi \leq k$, $\neg(\phi \leq k) = \phi > k$, $\neg\mu x.\phi = \mu x.\neg\phi\,[^{x}/_{\neg x}]$. Note

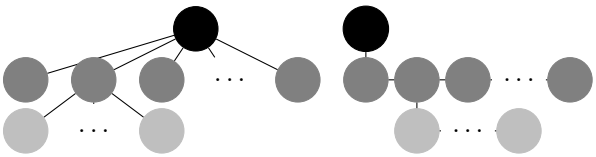

Fig. 2. Example of the bijection between n-ary and binary trees

that, for technical convenience, we consider an extension of formulas. This extension consists of *less than* counting formulas $\phi \leq k$ and the *true* formula $\top$ with the obvious semantics.

We require some notation before defining the Fischer-Ladner closure, from which we build the syntactic trees. Since integers associated to counting constraints are assumed to be in binary form, we thus define counter formulas as a boolean combination of propositions denoting an integer number. For example, for a sequence of propositions $p_1, p_2, \ldots$, the integer 1 is written $p_1 \wedge \bigwedge_{i>1} \neg p_i$, and the integer 5 (101 in binary) is written $p_3 \wedge \neg p_2 \wedge p_1 \wedge \bigwedge_{i>4} \neg p_i$. The amount of propositions required to define the counters of formula ϕ is bounded by $K(\phi)$, which is the sum of numerical constraints occurring in counting subformulas of ϕ. When clear from the context, we simply write K.

For a counting subformula $\phi \# k$ of a given formula a counter $\phi^{k'}$ set to k' is a sequence of fresh propositions occurring positively in the binary coding of the integer k'; and a flag $\phi^{\#k}$ is a fresh proposition. For instance, for the integer 5 coded as $c_2 \wedge \neg c_1 \wedge c_0$, we write ϕ^5 to denote c_2, c_0, where c_i are the corresponding propositions for the counting formula $\phi \# k$.

The Fischer-Ladner closure of a given formula is the set of its subformulas, together with their negation normal form, such that the fixed points are expanded once. Additionally, a counter and a flag for each counting subformula are also considered in the closure. All these information is obtained with the help of the relation R^{FL}, which is defined as follows.

Definition 6.1. We define the following binary relation R^{FL} over formulas for $i = 1, 2$:

$$
\begin{array}{ll}
R^{FL}(\phi, \mathsf{nnf}(\phi)), & R^{FL}(\phi_1 \wedge \phi_2, \phi_i), \\
R^{FL}(\phi_1 \vee \phi_2, \phi_i), & R^{FL}(\langle m \rangle \phi, \phi), \\
R^{FL}(\mu x.\phi, \phi\left[{}^{\mu x.\phi}/_{x}\right]), & R^{FL}(\phi \# k, \phi), \\
R^{FL}(\phi \# k, \phi^K), & R^{FL}(\phi \# k, \phi^{\#k}), \\
R^{FL}(\phi \# k, \psi), &
\end{array}
$$

where ψ is

$$\mu x_1.(\mu x_2.\phi \vee \langle \downarrow \rangle x_2 \vee \langle \rightarrow \rangle x_2) \vee \langle \uparrow \rangle x_1 \vee \langle \leftarrow \rangle x_1.$$

Notice that if ϕ is true in a model, then ψ is true in every node of the model. We use ψ to provide the necessary information for ϕ to navigate through the entire model.

We are now ready to define the Fischer-Ladner closure.

Definition 6.2 (Fischer-Ladner Closure)**.** The Fischer-Ladner closure of a given formula ϕ is defined as $\mathsf{CL}^{\mathsf{FL}}(\phi) = \mathsf{CL}^{\mathsf{FL}}(\phi)_k$, such that k is the smallest integer satisfying $\mathsf{CL}^{\mathsf{FL}}(\phi)_{k+1} = \mathsf{CL}^{\mathsf{FL}}(\phi)_k$, where $\mathsf{CL}^{\mathsf{FL}}(\phi)_0 = \{\phi\}$, $\mathsf{CL}^{\mathsf{FL}}(\phi)_{i+1} = \mathsf{CL}^{\mathsf{FL}}(\phi)_i \cup \{\psi' \mid R^{FL}(\psi, \psi'), \psi \in \mathsf{CL}^{\mathsf{FL}}(\phi)_i\}$.

The lean set of a given formula contains propositions, modal, and counting subformulas, together with counters and flags.

Definition 6.3 (Lean)**.** Given a formula ϕ and a proposition p' not occurring in ϕ, we define the lean as follows for all $m \in M$:

$$
\begin{aligned}
\mathsf{lean}(\phi) = &\{p, \langle m \rangle \psi, \psi \# k, \psi^K, \psi^{\#k} \in \mathsf{CL}^{\mathsf{FL}}(\phi)\} \\
&\cup \{\langle m \rangle \top, p'\}.
\end{aligned}
$$

The lean set contains all the required information to define tree nodes: propositions serve as labels, modal subformulas define the topology of the tree, and counters and flags serve to verify counting subformulas. p' is a proposition denoting the propositions not occurring in the input formula.

Consider the following formulas for $m \in \{\downarrow, \rightarrow, \uparrow, \leftarrow\}$: $\phi = [(p_1 > 1) \wedge p_2] > 4$, $\psi = (p_1 > 1) \wedge p_2$, $\phi_0 = \mu x.\psi \vee \bigvee_{\forall m} \langle m \rangle x$, and $\psi_0 = \mu x.p_1 \vee \bigvee_m \langle m \rangle x$.

The lean of ϕ is thus defined as follows for $m \in \{\downarrow, \rightarrow, \uparrow, \leftarrow\}$:

$$\begin{aligned}\text{lean}(\phi) =&\{p_1, p_2, \phi, p_1 > 1, \psi^7, p_1^7, \psi^{>4}, p_1^{>1},\\ &\langle m\rangle\phi_0, \langle m\rangle\psi_0, p', \langle m\rangle\top\},\end{aligned}$$

$K = 7$. Now recall that ϕ^7 denote 3 propositions that serve to express the binary coding of the integers from 0 to 7.

We are now ready to define the syntactic notion of tree nodes.

Definition 6.4 (ϕ-Nodes)**.** Given a formula ϕ, a ϕ-node n^ϕ is defined as a subset of lean(ϕ), such that:

- at least one proposition of ϕ occurs;
- if $\langle m\rangle\psi$ occurs, then $\langle m\rangle\top$ also does;
- both $\langle\leftarrow\rangle\top$ and $\langle\uparrow\rangle\top$ can not occur;
- counting formulas are always present;
- exactly one counter for each counting formula is present, i.e., if $\phi\#k \in n^\phi$, then $\phi^{k'} \in n^\phi$;
- counters must be consistent with counting formulas and flags, i.e., $\psi^{k_0}, \psi \leq k \in n$ if and only if $k_0 \leq k$, and $\psi^{k_0}, \psi^{>k} \in n$ if and only if $k_0 > k$.

The set of ϕ-nodes is written N^ϕ. If the context is clear, we often call a ϕ-node simply a node, and we write n instead of n^ϕ.

We now define trees as triples (n, X_1, X_2), where n is the root of the tree and X_1 and X_2 are the respective left and right subtrees. Given a formula, a Fischer-Ladner tree, or simply a tree, is inductively defined as follows: the empty set $\emptyset$ is a tree; (n^ϕ, X_1, X_2) is also a tree, provided that X_1 and X_2 are also trees.

Consider $\phi, \psi, \phi_0, \psi_0$ from the example above. We define the following syntactic tree model for ϕ:

$$\begin{aligned}T =&(n_0, (n_1, (n_3, \emptyset, \emptyset), (n_4, \emptyset, \emptyset)), (n_2, (n_5, \emptyset, \emptyset),\\ &(n_6, \emptyset, \emptyset)),\end{aligned}$$

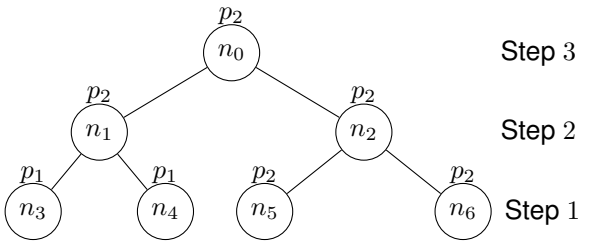

Fig. 3. Fischer-Ladner tree model for $\phi = [(p_1 > 1) \wedge p_2] > 4$

where

$$\begin{aligned}n_0 =&\{p_2, \phi, p_1 > 1, p_1^2, p_1^{>1}, \psi^5, \psi^{>4},\\ &\langle\downarrow\rangle\psi_0, \langle\rightarrow\rangle\psi_0, \langle\downarrow\rangle\phi_0, \langle\rightarrow\rangle\phi_0, \langle\downarrow\rangle\top, \langle\rightarrow\rangle\top\},\\ n_1 =&\{p_2, \phi, p_1 > 1, p_1^2, p_1^{>1}\psi^1, \langle\downarrow\rangle\psi_0, \langle\rightarrow\rangle\psi_0,\\ &\langle\uparrow\rangle\psi_0, \langle\downarrow\rangle\phi_0, \langle\rightarrow\rangle\phi_0, \langle\uparrow\rangle\phi_0, \langle\downarrow\rangle\top, \langle\rightarrow\rangle\top, \langle\uparrow\rangle\top\},\\ n_2 =&\{p_2, \phi, p_1 > 1, p_1^2, p_1^{>1}, \psi^3, \langle\downarrow\rangle\psi_0, \langle\rightarrow\rangle\psi_0, \langle\leftarrow\rangle\psi_0,\\ &\langle\downarrow\rangle\phi_0, \langle\rightarrow\rangle\phi_0, \langle\leftarrow\rangle\phi_0, \langle\downarrow\rangle\top, \langle\rightarrow\rangle\top, \langle\leftarrow\rangle\top\},\\ n_3 =&\{p_1, \phi, p_1 > 1, p_1^1, \langle\uparrow\rangle\phi_0, \langle\uparrow\rangle\psi_0, \langle\uparrow\rangle\top\},\\ n_4 =&\{p_1, \phi, p_1 > 1, p_1^1, \langle\leftarrow\rangle\phi_0, \langle\leftarrow\rangle\psi_0, \langle\leftarrow\rangle\top\},\\ n_5 =&\{p_2, \phi, p_1 > 1, \psi^1, \langle\uparrow\rangle\phi_0, \langle\uparrow\rangle\psi_0, \langle\uparrow\rangle\top\},\\ n_6 =&\{p_2, \phi, p_1 > 1, \psi^1, \langle\leftarrow\rangle\phi_0, \langle\leftarrow\rangle\psi_0, \langle\leftarrow\rangle\top\}.\end{aligned}$$

Fig. 3 depicts a graphical representation of T.

The satisfiability algorithm, described in Algorithm 1, builds candidate trees in a bottom-up manner: iteratively, starting from leaf nodes, we check at each step if the input formula is satisfied by candidate trees, in case the formula is not satisfied, we consistently add parents to previously built trees. The algorithm returns 1 if a satisfying tree is found. In case a satisfying tree could not be found, and no more candidate trees can be built, then the algorithm returns 0.

Consider the formula ϕ defined in the example above. Then the Fischer-Ladner tree, as also defined above, is built by the algorithm in 3 steps. In the first step, all the leaves are considered, that is, nodes without children, such that the counters are properly initialized (Definition 6.6). It is then easy to see that n_3, n_4, n_5, n_6 are all leaves. Since p_1 is occurring in both, n_3 and n_4, then the counter p_1^1 is

Algorithm 1 Satisfiability Algorithm

$Y \leftarrow N^{\phi}$
$\mathcal{X} \leftarrow Leaves(Y)$
$\mathcal{X}_0 \leftarrow \emptyset$
while $\mathcal{X} \neq \mathcal{X}_0$ **do**
  if $\mathcal{X} \Vdash \phi$ **then**
    return 1
  end if
  $\mathcal{X}_0 \leftarrow \mathcal{X}$
  $(\mathcal{X}, Y) \leftarrow Update(\mathcal{X}, Y)$
end while
return 0

also in the same nodes. Since both p_2 and $p_1 > 1$ are in n_5 and n_6, then $\psi = p_2 \wedge p_1 > 1$ is true in both nodes, and consequently ψ^1 is also in n_5 and n_6. However, none of the leaves satisfies ϕ, then, in the second step, n_1 is added as parent to both n_3 and n_4. n_2 is also added as parent to n_5 and n_6. Since ψ is true in n_1 and n_2, then the counter for ψ is incremented in both nodes resulting that in n_1 we have ψ^1, and in n_2 we have ψ^3. However, none of the trees built in step 2 satisfies ϕ. In step 3, n_0 is then added as parent of n_1 and n_2. Since ψ holds in n_0, then we update the counter to ψ^5, and ϕ is then finally satisfied. The resulting tree of this process is depicted in Fig. 3. Each step corresponds to one level of the tree.

We now provide a precise description of the algorithm components. If a tree T is a model for a formula ϕ, it is said that T satisfies (entails) ϕ.

Definition 6.5. The entailment of a formula by a node is defined by

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

The entailment relation is now extended for trees and formulas. A formula ϕ is satisfied by a tree X, written $X \Vdash \phi$ if and only if there is a node n in X, such that $n \vdash \phi$; formulas of the forms $\langle\uparrow\rangle\psi$ and $\langle\leftarrow\rangle\psi$ do not occur in the root of X; and all the flags are in the root. A set of trees $\mathcal{X}$ entails a formula ϕ, written $\mathcal{X} \Vdash \phi$ if and only if there is a tree X in $\mathcal{X}$ s.t. $X \Vdash \phi$. The relation $\nVdash$ is defined as expected.

The set of leaves contains nodes without children. In the leaves, counters are also properly initialized.

Definition 6.6 (Leaves)**.** Given a set of nodes X, the set of leaves is defined as follows:

$$Leaves(X) = \{(n, \emptyset, \emptyset) \mid n \in X, \langle\downarrow\rangle\phi, \langle\rightarrow\rangle\phi \notin n, \left[(\phi^1 \in n, n \vdash \phi) \text{ or } (\phi^0 \in n, n \nvdash \phi)\right].\}$$

A node n containing a modal formula $\langle m\rangle\psi$ can be linked to another node n' through a modality m if and only if there is a witness of ψ in n', that is, $n' \vdash \psi$. This notion is defined by the relation Δ_m.

Definition 6.7. Given two nodes n_1, n_2 and formula ϕ, we say that the nodes are modally consistent with respect to the formula $\Delta_m(n_1, n_2)$ for $m \in \{\downarrow, \rightarrow\}$ if and only if for all formulas $\langle m\rangle\psi_1, \langle\overline{m}\rangle\psi_2 \in$ lean(ϕ), we have that

— $\langle m\rangle\psi_1 \in n_1$ if and only if $n_2 \vdash \psi_1$, and

— $\langle\overline{m}\rangle\psi_2 \in n_2$ if and only if $n_1 \vdash \psi_2$.

When adding parents, it is also necessary to ensure that counting formulas are satisfied. Recall that, according to the definition of ϕ-nodes, counting formulas and flags are consistent with counters. It is then only required to update the counters and to copy the flags that are already in the subtrees. We have two cases. The first one is when we add a parent to both, a left and a right subtrees. The second case is when a parent is added to one subtree only. Consider the first case.

Definition 6.8. It is said that three nodes n_0, n_1, n_2 are consistent with respect to their counters, denoted by $\#(n_0, n_1, n_2)$ if and only if

— $\psi^{k_0} \in n_0$ and $n_0 \vdash \psi$ if and only if $\psi^{k_1} \in n_1$, $\psi^{k_2} \in n_2$ and $k_0 = k_1 + k_2 + 1$ if $k_0 \leq K$, otherwise $k_0 = K$;

— $\psi^{k_0} \in n_0$ and $n_0 \nvdash \psi$ if and only if $\psi^{k_1} \in n_1$, $\psi^{k_2} \in n_2$ and $k_0 = k_1 + k_2$ if $k_0 \leq K$, otherwise $k_0 = K$; and

— if $\psi^{>k} \in n_i$ for any $i \in \{1, 2\}$, then $\psi^{>k} \in n_0$.

The second case ($\#(n_0, n_i)$) is defined in an analogous manner.

Recall that the $Update$ function is used to consistently add parents to previously built trees. Now, with the notions of modal and counter consistency (Definitions 6.7 and 6.8) already defined, we are now ready to give a precise description of the $Update$ function.

Definition 6.9. Given a set of trees $\mathcal{X}$ and a set of nodes Y, the function $Update(\mathcal{X}, Y)$ is defined as the tuple $(\mathcal{X}', Y')$, such that

— $\mathcal{X}' = \{(n, X_\downarrow, X_\rightarrow) \mid n \in Y, X_i \in \mathcal{X}, \Delta_i(n, n_i), \#(n, n_1, n_2)\}$, where $i = \downarrow, \rightarrow$ and n_i is the root of X_i; or

— $\mathcal{X}' = \{(n, X_\downarrow, X_\rightarrow) \mid n \in Y, X_i \in \mathcal{X}, \Delta_i(n, n_i), \#(n, n_i)\}$ in case $X_j = \emptyset$ with $i \neq j$; and

— $Y' = Y \setminus \{n\}$.

It is easy to see that the algorithm has a finite number of steps if we notice that the number of nodes is finite and that the $Update$ function is monotone.

In order to show that the algorithm is correct, we then prove it to be sound and complete.

Theorem 6.1 (Soundness)**.** *If the algorithm returns* 1 *for the input formula* ϕ*, then there is a tree model satisfying* ϕ*.*

Proof. By assumption, there is a triple X such that $X \Vdash \phi$. We will now construct a tree model T from X.

— The set of propositions P which are the ones in lean(ϕ).

— The nodes of T are N^ϕ.

— We now define the edges of T. For every triple (n, X_1, X_2) of X, we define $\mathcal{R}(n, \downarrow) = n_1$ and $\mathcal{R}(n, \rightarrow) = n_2$, provided that n_1 and n_2 are the respective roots of X_1 and X_2.

— We label the nodes in an obvious manner: if $p \in n$, then $p \in L(n)$.

It is now shown by structural induction on ϕ that T satisfies ϕ. All cases are straightforward. For the case of fixed-point subformulas, recall that there is an equivalent finite unfolding, that is, $\mu x.\psi \equiv \phi\left[{}^{\mu x.\phi}/_{x}\right]$. □

For completeness it is assumed that there is a satisfying tree $\mathcal{T}$ for the formula ϕ, and then it is shown that the algorithm returns 1. The proof comes in two steps: we first construct an equivalent lean labeled version of $\mathcal{T}$, and then we show that the algorithm can actually construct such a lean labeled tree.

Definition 6.10. Given a satisfying tree $\mathcal{T}$ of a formula ϕ, we define its lean version $X^\mathcal{T}$ as follows: $X^\mathcal{T}$ has the same nodes and shape than $\mathcal{T}$; each node n in $X^\mathcal{T}$ is labeled with the formulas ψ in lean(ϕ) such that n in $\mathcal{T}$ satisfies ψ, and the labels corresponding to the counters are pinned up in a similar manner as the algorithm does, that is, in an increasing bottom-up order (with bound K) in the tree.

Lemma 6.1. *If a tree* $\mathcal{T}$ *satisfies a formula* ϕ*, then* ϕ *is entailed by* $X^\mathcal{T}$*.*

Proof. We proceed by induction on the derivation of $n \vdash \phi$. Most cases are immediate by induction and the construction of X^T.

For the fixpoint case $\mu x.\psi$, we test $\psi\left[{}^{\mu x.\psi}/_{x}\right]$. We then proceed by structural induction again. This is also straightforward since variables, and hence unfolded fixed-points, can only occur in the scope of a modality or a counting formula. □

One crucial point in the completeness proof is to show that N^ϕ contains enough nodes to satisfy ϕ. It is well-known that the standard Fischer-Ladner construction of models provides the required amount of nodes for simple μ-calculus formulas without counting [6]. Since counting subformulas impose bounds on the number of certain nodes, it may be required to duplicate ϕ-nodes. Counters are then introduced in the Fischer-Ladner construction in order to distinguish potentially identical nodes. We now show that counters are introduced in a consistent manner.

Lemma 6.2. *Given a satisfying tree $\mathcal{T}$ of a formula ϕ, there is a tree entailing ϕ, such that for every path from its root to a leave, there are no identical ϕ-nodes.*

Proof. If every path in X^T does not contain identical nodes, then we are done.

Consider now the case when we have two identical nodes n_1 and n_2 in a path of X^T. Without loss of generality, we assume that n_1 is above n_2. We then proceed to build a tree X from X^T, such that n_2 is grafted upon n_1. That is, the path between n_1 and n_2 is removed, not including n_1 but including n_2. n_1 is then linked to the subtrees of n_2. X can then be seen as the pruned version of X^T.

We now show that X also entails ϕ by induction on the derivation of $X \vdash \phi$. Most cases are immediate by the construction of X and by induction.

Consider now the case of counting subformulas. Since these subformulas are true in every node, then the only important thing is to be sure that the counted nodes are not part of the pruned path. This is not possible since the counters in n_2 are the same as the ones in n_1, that is, the counters are not increased between n_1 and n_2. □

Now that we know how the tree is composed (Lemma 6.1), and that we have enough copies of nodes to construct it (Lemma 6.2), completeness follows.

Theorem 6.2 (Completeness)**.** *If a formula ϕ is satisfiable, then the algorithm returns 1.*

As in [2, 7], the time complexity of the satisfiability algorithm is single exponential on the number of nodes (automaton states) introduced by the Fischer-Ladner construction.

Theorem 6.3 (Complexity)**.** *μTLIN satisfiability is in EXPTIME-complete.*

Proof. Notice first that the size of the lean is at most polynomial with respect to the formula size. We then show that the complexity of the algorithm is at most exponential with respect to the lean size.

Now note that the size of N^ϕ is exponentially bounded by the lean size. Then, in the loop there is at most an exponential number of steps.

Computing the set $Leaves$ takes exponential time since N^ϕ is traversed once.

Now note that testing the relation $\vdash$ costs linear time with respect to the size of the node. Then the entailments $\Vdash$ and $\nVdash$ take at most exponential time.

The $Update$ function costs at most exponential time by the following facts: traversals on $\mathcal{X}$ and Y take exponential time; and the costs of the relations Δ and $\#$ are linear. Since each step in the loop takes at most exponential time, we conclude that the overall complexity is single exponential. The lower bound comes from the fact that the logic can encode all finite tree automata and is closed under negation, then satisfiability is hard for EXPTIME, and hence complete. □

Recall that regular path queries (XPath) and regular tree expressions (XML schemas), extended with counting constructs, can be encoded in terms of logical formulas with linear size with respect to the original queries and types (Theorems 4.1 and 5.1). We can then conclude that the logic can be used as an optimal query reasoning framework for XML trees.

Corollary 6.1. *The emptiness, containment, and equivalence of regular queries with counting and regular tree expressions with interleaving and counting are decidable in EXPTIME.*

7 Conclusions

In this paper we introduced an extension of the fully enriched μ-calculus for trees with global numerical constraints. These constraints express numerical restrictions on the number of nodes in any tree region. We also showed that an extension of regular path queries (XPath) with counting constructs can be efficiently expressed in terms of logic formulas. The same applies for an extension of regular tree languages (XML schemas) with counting and interleaving operators. In addition, we provided a satisfiability algorithm for the logic with EXPTIME complexity. These results, together with the fact that the logic is closed under negation, implies an optimal reasoning framework for counting extensions of XML queries and schemas.

Regarding further research perspectives, the implementation of the satisfiability algorithm with the aim of Binary Decision Diagrams promises immediate relevance in the static analysis of XML applications. We believe that extensions of XPath queries with data value tests [11] can also be studied in the context of expressive modal logics. Furthermore, we believe that our logic can also be used in the verification of balanced tree structures, such as AVL, red-black, and splay trees [20].

References

1. **Bárcenas, E.** (**2011**). *Raisonnement automatisè sur les arbres avec des contraintes de cardinalité*. Ph.D. thesis, Université de Grenoble.
2. **Bárcenas, E., Genevès, P., Layaïda, N., & Schmitt, A.** (**2011**). Query reasoning on trees with types, interleaving, and counting. **Walsh, T.**, editor, *IJCAI*, IJCAI/AAAI, pp. 718–723.
3. **Bárcenas, E. & Lavalle, J.** (**2013**). Expressive reasoning on tree structures: Recursion, inverse programs, presburger constraints and nominals. **Castro, F., Gelbukh, A. F., & González, M.**, editors, *MICAI (1)*, volume 8265 of *Lecture Notes in Computer Science*, Springer, pp. 80–91.
4. **Bárcenas, E. & Lavalle, J.** (**2014**). Global numerical constraints on tress. *Logical Methods in Computer Science*, Vol. 10, No. 2.
5. **Bianco, A., Mogavero, F., & Murano, A.** (**2009**). Graded computation tree logic. *LICS*, IEEE Computer Society, pp. 342–351.
6. **Bonatti, P. A., Lutz, C., Murano, A., & Vardi, M. Y.** (**2006**). The complexity of enriched μ-calculi. **Bugliesi, M., Preneel, B., Sassone, V., & Wegener, I.**, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, Springer, pp. 540–551.
7. **Calvanese, D., De Giacomo, G., Lenzerini, M., & Vardi, M. Y.** (**2010**). Node selection query languages for trees. **Fox, M. & Poole, D.**, editors, *AAAI*, AAAI Press.
8. **Dal-Zilio, S., Lugiez, D., & Meyssonnier, C.** (**2004**). A logic you can count on. **Jones, N. D. & Leroy, X.**, editors, *POPL*, ACM, pp. 135–146.
9. **Demri, S. & Lugiez, D.** (**2006**). Presburger modal logic is PSPACE-Complete. **Furbach, U. & Shankar, N.**, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, Springer, pp. 541–556.
10. **Ferrante, A., Napoli, M., & Parente, M.** (**2009**). Graded-CTL: Satisfiability and symbolic model checking. **Breitman, K. & Cavalcanti, A.**, editors, *ICFEM*, volume 5885 of *Lecture Notes in Computer Science*, Springer, pp. 306–325.
11. **Figueira, D., Figueira, S., & Areces, C.** (**2014**). Basic model theory of xpath on data trees. **Schweikardt, N., Christophides, V., & Leroy, V.**, editors, *ICDT*, OpenProceedings.org, pp. 50–60.
12. **Gelade, W., Martens, W., & Neven, F.** (**2009**). Optimizing schema languages for XML: Numerical constraints and interleaving. *SIAM J. Comput.*, Vol. 38, No. 5, pp. 2021–2043.
13. **Genevès, P., Layaïda, N., & Schmitt, A.** (**2007**). Efficient static analysis of XML paths and types. **Ferrante, J. & McKinley, K. S.**, editors, *PLDI*, ACM, pp. 342–351.
14. **Gruber, H. & Holzer, M.** (**2009**). Tight bounds on the descriptional complexity of regular expressions. **Diekert, V. & Nowotka, D.**, editors, *Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, Springer, pp. 276–287.
15. **Henriksen, J., Jensen, J., Jørgensen, M., Klarlund, N., Paige, B., Rauhe, T., & Sandholm, A.** (**1995**). Mona: Monadic second-order logic in practice. *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*.
16. **Hosoya, H., Vouillon, J., & Pierce, B. C.** (**2005**). Regular expression types for XML. *ACM Trans. Program. Lang. Syst.*, Vol. 27, No. 1, pp. 46–90.
17. **Janin, D. & Walukiewicz, I.** (**1996**). On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. **Montanari, U. & Sassone, V.**, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, Springer, pp. 263–277.
18. **Kilpeläinen, P. & Tuhkanen, R.** (**2003**). Regular expressions with numerical occurrence indicators - preliminary results. **Kilpeläinen, P. & Päivinen, N.**, editors, *SPLST*, University of Kuopio, Department of Computer Science, pp. 163–173.
19. **Lipshitz, L.** (**1976**). The diophantine problem for addition and divisibility. *Transaction of the American Mathematical Society*, Vol. 235, pp. 271–283.
20. **Manna, Z., Sipma, H. B., & Zhang, T.** (**2007**). Verifying balanced trees. **Artëmov, S. N. & Nerode, A.**, editors, *LFCS*, volume 4514 of *Lecture Notes in Computer Science*, Springer, pp. 363–378.

21. **Marx, M.** (**2005**). Conditional XPath. *ACM Trans. Database Syst.*, Vol. 30, No. 4, pp. 929–959.
22. **Murata, M., Lee, D., Mani, M., & Kawaguchi, K.** (**2005**). Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, Vol. 5, No. 4, pp. 660–704.
23. **Seidl, H., Schwentick, T., & Muscholl, A.** (**2003**). Numerical document queries. *PODS*, ACM, pp. 155–166.
24. **Seidl, H., Schwentick, T., Muscholl, A., & Habermehl, P.** (**2004**). Counting in trees for free. **Díaz, J., Karhumäki, J., Lepistö, A., & Sannella, D.**, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, Springer, pp. 1136–1149.
25. **ten Cate, B. & Marx, M.** (**2009**). Axiomatizing the logical core of XPath 2.0. *Theory Comput. Syst.*, Vol. 44, No. 4, pp. 561–589.

Everardo Bárcenas is a researcher at the Consejo Nacional de Ciencia y Tecnología (CONACYT). He graduated from Université de Grenoble, France. His research interests include automated reasoning, modal logics, knowledge representation, and formal methods.

Article received on 09/07/2014; accepted on 21/04/2015.
Corresponding author is Everardo Bárcenas.