ISSN 2007-9737

Presburger Constraints in Trees

Everardo Bárcenas[1], Edgard Benítez-Guerrero[2], Jesús Lavalle[3], Guillermo Molero-Castillo[1]

[1] Universidad Nacional Autónoma de México,
Mexico

[2] Universidad Veracruzana,
Mexico

[3] Benemérita Universidad Autónoma de Puebla
Mexico

ebarcenas@unam.mx, edbenitez@uv.mx, jlavalle@cs.buap.mx, gmoleroca@fi-b.unam.mx

Abstract. The fully enriched μ-calculus is an expressive propositional modal logic with least and greatest fixed-points, nominals, inverse programs and graded modalities. Several fragments of this logic are known to be decidable in EXPTIME. However, the full logic is undecidable. Nevertheless, it has been recently shown that the fully enriched μ-calculus is decidable in EXPTIME when its models are finite trees. In the present work, we study the fully-enriched μ-calculus for trees extended with Presburger constraints. These constraints generalize graded modalities by restricting the number of children nodes with respect to Presburger arithmetic expressions. We show that this extension is decidable in EXPTIME. In addition, we also identify decidable extensions of regular tree languages (XML schemas) with interleaving and counting operators. This is achieved by a linear characterization in terms of the logic. Regular path queries (XPath) with Presburger constraints on children paths are also characterized. These results imply new optimal reasoning (emptiness, containment, equivalence) bounds on counting extensions of XPath queries and XML schemas.

Keywords. Presburger arithmetic, modal logics, automated reasoning, XPath, regular languages, interleaving.

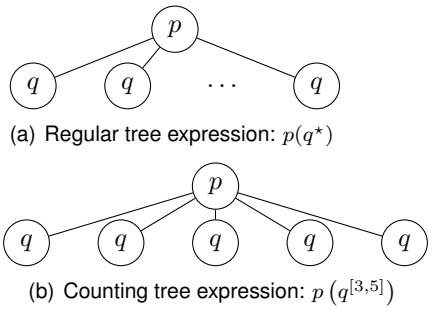
1 Introduction

The μ-calculus is an extension of the propositional modal logic with least and greatest fixed-points. This logic subsumes many temporal, modal and description logics (DLs), such as the Propositional Dynamic Logic (PDL) and the Computation Tree Logic (CTL) [10, 9]. Due to its expressive power and nice computational properties, the μ-calculus has been extensively used in many areas of computer science, such as program verification, concurrent systems and knowledge representation. In this last domain, the μ-calculus has been particularly useful in the identification of expressive and computationally well-behaved DLs [10], which are used as the web ontology language OWL, now a standard technology for the W3C. Another standard for the W3C is the XPath query language for XML.

XPath also takes an important role in many XML technologies, such as XProc, XSLT and XQuery. Due to its capability to express recursive and multi-directional navigation, the μ-calculus has also been successfully used as a framework for the evaluation and reasoning of XPath queries [5, 4, 12]. However, extending (Presburger) arithmetical constraints for XPath queries leads to undecidablity [36]. In the current paper, we identify a decidable extension, vía a logic characterization, of XPath queries with Presburger arithmetical constraints on children paths.

The μ-calculus is as expressive as the monadic second order logic (MSO) [12], it has thus been successfully used in the XML setting in the

ISSN 2007-9737



(a) Regular tree expression: $p(q^\star)$

(b) Counting tree expression: $p\left(q^{[3,5]}\right)$

Fig. 1. Tree expressions

description of schema languages [4], which can be seen as the tree version of regular expressions. Analogously as regular expressions are interpreted as sets of strings, XML schemas (regular tree expressions) are interpreted as sets of unranked trees (XML documents). For example, expression $p(q^\star)$ represents the sets of trees rooted at p with either none, one or more children subtrees matching q. See Figure 1(a) for an interpretation of $p(q^\star)$. Counting operators impose occurrence bounds on children [28]. For instance, $p\left(q^{[3,5]}\right)$ denotes the trees rooted at p, with at least 3 but no more than 5 children matching q. In Figure 1(b), it is depicted an interpretation of $p\left(q^{[3,5]}\right)$. In the present work, we proposea new EXPTIME decision algorithm for regular tree languages with counting operators. Furthermore, we also identify a new extension of regular tree languages with the interleaving operator [27]. This extension has also an EXPTIME upper bound. This operator is used to denote the trees resulting from the permutation of siblings, and it has also applications in algebraic approaches of concurrent computation models. Also, in XML schema languages there are interleaving operators, such as in RelaxNG.

1.1 Related Work

The extension of the μ-calculus with nominals, inverse programs and graded modalities is known as the fully enriched μ-calculus [10]. Nominals are intuitively interpreted as singleton sets, inverse programs are used to express past properties (backward navigation along accesability relations), and graded modalities express numerical constraints on the number immediate successor nodes [10]. However, satisfiability/validity of the fully enriched μ-calculus was proven by Bonatti and Peron to be undecidable [11]. Nevertheless, Bárcenas et al. [4] recently showed that the fully enriched μ-calculus is decidable in single exponential time when its interpretations are restricted to finite trees. Graded modalities in the context of trees are used to constrain the number of children nodes with respect to natural numbers. In this work, we introduce a generalization of graded modalities.

This generalization considers numerical bounds on children with respect to Presburger arithmetical expressions, as for instance $\phi > \psi$, which restricts the number of children where ϕ holds to be strictly greater than the number of children where ψ is true. Other works have previously considered Presburger constraints on tree logics. MSO with Presburger constraints was shown to be undecidable by Seidl et al. [31]. Demri and Lugiez proved a PSPACE-complete bound on the propositional modal logic with Presburger constraints [16]. A tree logic with a fixed-point and Presburger constraints was shown to be decidable in EXPTIME by Seidl et al. [33, 32]. In the current work, we push decidability further by allowing nominals and inverse programs in addition to Presburger constraints.

Regarding expressiveness, in [17] is proposed an extension of MSO with counting on unranked trees, together with the corresponding weighted automata. Other forms of counting considering more extensive tree regions, such as ancestors or descendants, have been studied in [37, 2, 8, 39, 5, 34, 1, 24], however in contrast with the current work, these works consider node occurrence constraints with respect to natural numbers only. An extension of first order logic with two variables FO^2 with counting quantifiers interpreted over two forests of ranked trees was recently proposed in [13, 14]. The counting quantifiers consist of existential $\exists^{\#k}$ ($\# \in \{\leq, \geq, =\}$) restrictions with respect to a constant k. This logic was shown to be in NEXPTIME.

As in this article, Fischer-Ladner satisfiability algorithms for counting extensions of the μ-calculus for trees are introduced in [4, 5].

In these works, it was also showed that these counting constraints with respect to numbers only, although exponentially more succinct, does not introduce extra-expressive power. This allowed to use the traditional Fixed-Point Theorem for the μ-calculus to prove the algorithm correctness. Contrastingly, in the current work we propose a counting extension of the μ-calculus with full Presburger arithmetic, which clearly implies more expressive power. In order to prove the algorithm correctness, we prove a generalization of the Fixed-Point Theorem for the Presburger extension of the μ-calculus for trees, which is also technical result of independent interest.

Complexity and succinctness of regular tree (string) languages extended with counting and interleaving operators have been extensively studied in [28, 27, 19, 4, 15]. In [19], Gelade shows that the interleaving operator is exponentially more succinct, even when it is directly encoded by tree automata. It is also shown that hardcoding counting operators produces doubly exponential larger expressions. Meyer and Stockmeyer showed in [28] that the equivalence of regular expressions with counting operators is EXPSPACE-complete. EXPSPACE completeness of the equivalence of regular expressions with interleaving was proven in [27].

In [4], it is described and extension of regular trees expressions with counting and interleaving, where emptiness, containment and equivalence are decidable in EXPTIME. In the current work, we identify further extensions of interleaving occurring in recursive and disjunctive fragments. Emptiness, containment and equivalence are also proved to be EXPTIME.

Regarding counting, operators introduced in [4], although impose occurrence bounds on children, do not restrict the consecutive occurrence of children. This contrasts with traditional semantics of counting in regular languages [28, 19]. In the current work, by means of Presburger constraints, we identify an extension of regular tree expressions with traditional counting operators decidable in EXPTIME. In [15], it is introduced a polynomial algorithm for the containment of regular trees with both counting and interleaving.

The algorithm assumes two main restrictions on tree expressions: propositions occur only once, and counting can be applied to propositions only. The EXPTIME algorithm proposed in the current work overcomes these two limitations.

Counting operators on regular paths (XPath) have been studied before in [36, 4, 5]. ten Cate and Marx [36] showed that Presburger constraints on full regular paths lead to an undecidable formalism.

In [4], it was shown that when restricting only children with respect to a natural number (encoded in binary), reasoning (emptiness, containment and equivalence) is in EXPTIME. This result was extended to operators capable of constraining (w.r.t. a binary number) any regular path, including ancestors, descendants, compositions, etc. [5]. In this paper, we show that full Presbuger arithmetic becomes decidable on children paths. Furthermore, we set new optimal EXPTIME bounds for containment and equivalence on full (multi-directional) regular paths with Presburger constraints on children.

1.2 Outline

We introduce a modal logic for trees with fixed-points, inverse programs, and Presburger constraints in Section 2.

In Section 3, an EXPTIME satisfiability algorithm is described and proved correct. Also, it is shown that the computational cost of the algorithm is single exponential time with respect to the size of the input formula, even if the Presburger constraints are encoded in binary form.

In Section 4, we introduce extensions of regular tree languages with counting and interleaving operators.

In Section 5, regular path queries extended with Presburger constraints on children are introduced. Both extensions of regular trees and paths are shown to be succinctly captured in terms of logic formulas. This implies the satisfiability algorithm can be used as an optimal reasoning framework for regular trees and paths with counting and interleaving.

A summary together with a discussion of further research perspectives are reported in Section 6.

ISSN 2007-9737

2 Tree Logic with Recursion, Inverse, and Presburger Constraints

In this section, we introduce an expressive modal logic for finite unranked tree models. The tree logic (TL) is equipped with operators for recursion (μ), inverse programs (I), and Presburger constraints (C). This logic can be seen as the fully enriched μ-calculus [10], extended with Presburger constraints, interpreted over tree structures.

2.1 Syntax and Semantics

We first consider a set of propositions P, a finite set of modalities $M = \{\downarrow, \rightarrow, \uparrow, \leftarrow\}$, and a set of variables X.

Definition 1 (μTLIC syntax)**.** The set of μTLIC formulas is defined by the following grammar:

$$\phi := p \mid x \mid \neg\phi \mid \phi \vee \phi \mid \langle m \rangle \phi \mid \mu x.\phi \mid \gamma > b,$$
$$\gamma := a\phi \mid \gamma + \gamma,$$

where $p \in P$, $x \in X$, $m \in M$, $b \in \mathbb{N}$, $k \in \mathbb{N} \setminus \{0, 1\}$, $a \in \mathbb{Z} \setminus \{0\}$. Numbers a, k and b are asumed to be encoded in binary. We consider the following assumptions about variable occurrences: as usual, in order to ensure the existence of fixed-points, we assume variables occurs positively, that is, under the scope of an even number of negations [38]; also, variables are guarded, that is, variables occur bounded (where μ is the only binding operator) and under the scope of a modal formula ($\langle m \rangle \phi$), or a counting formula ($\gamma > b$) [38]; and in order to make the least and greatest fixed points coincide, we assume variables are cycle-free, that is, variables does not occur under the scope of a modality and its converse [20].

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;
- the set of nodes $\mathcal{N}$ is defined as a complete prefix-closed non-empty finite set of words over the natural numbers $\mathbb{N}$, that is, $\mathcal{N}$ is a finite set of words $\mathcal{N} \subseteq \mathbb{N}^\star$, such that if $n \cdot i \in \mathcal{N}$, where $n \in \mathbb{N}^\star$ and $i \in \mathbb{N}$, then also $n \in \mathcal{N}$;
- $\mathcal{R} : \mathcal{N} \times M \times \mathcal{N}$ is a transition relation, written $n \in \mathcal{R}(n', m)$, such that for all $(n \cdot i), (n \cdot i+1) \in \mathcal{N}$, where $i \in \mathbb{N}$, $n \in \mathcal{R}(n \cdot i, \downarrow)$, $(n \cdot i) \in \mathcal{R}(n, \uparrow)$, $(n \cdot i+1) \in \mathcal{R}(n \cdot i, \rightarrow)$ and $(n \cdot i) \in \mathcal{R}(n \cdot i+1, \leftarrow)$, we say n is the parent of $n \cdot i$, hence $n \cdot i$ is the child of n, $n \cdot i+1$ is a following (right) sibling of $n \cdot i$, and hence $n \cdot i$ is a previous (left) sibling of $n \cdot i + 1$; and
- $L : \mathcal{N} \times P$ is a left-total labeling relation, written $p \in L(n)$.

In the setting of XML documents (unranked trees), node labels are defined by a function instead of a relation, that is, exactly one proposition holds at each node. The satisfiability algorithm described in Section 3 can easily be adapted for this restriction (see Definition 6).

Given a tree structure, a valuation V of variables is defined as a function from the set variables X to a set of nodes $V : X \mapsto 2^{\mathcal{N}}$. For nodes $\mathcal{N}' \subseteq \mathcal{N}$, we write $V\left[{}^{\mathcal{N}'}/_{x}\right]$ to denote the valuation V', such that $V'(x) = \mathcal{N}'$ and $V'(x') = V(x)$ for $x' \neq x$.

Definition 2 (μTLIC semantics)**.** Given a tree structure $\mathcal{T}$ and a valuation V, μTLIC formulas are interpreted as follows:

$$\begin{aligned}
[\![p]\!]^{\mathcal{T}}_V &= \{n \mid p \in L(n)\}, \\
[\![x]\!]^{\mathcal{T}}_V &= V(x), \\
[\![\neg\phi]\!]^{\mathcal{T}}_V &= \mathcal{N} \setminus [\![\phi]\!]^{\mathcal{T}}_V, \\
[\![\phi \vee \psi]\!]^{\mathcal{T}}_V &= [\![\phi]\!]^{\mathcal{T}}_V \cup [\![\psi]\!]^{\mathcal{T}}_V, \\
[\![\langle m \rangle \phi]\!]^{\mathcal{T}}_V &= \left\{ n \mid \mathcal{R}(n, m) \cap [\![\phi]\!]^{\mathcal{T}}_V \neq \emptyset \right\}, \\
[\![\mu x.\phi]\!]^{\mathcal{T}}_V &= \bigcap \left\{ \mathcal{N}' \subseteq \mathcal{N} \;\middle|\; [\![\phi]\!]^{\mathcal{T}}_{V\left[{}^{\mathcal{N}'}/_{x}\right]} \subseteq \mathcal{N}' \right\}, \\
[\![\gamma > b]\!]^{\mathcal{T}}_V &= \left\{ n \mid \|\gamma\|^{\mathcal{T}_n}_V > k \right\}, \\
\|a\phi\|^{\mathcal{T}_n}_V &= a \times \left| \mathcal{R}(n, \downarrow) \cap [\![\phi]\!]^{\mathcal{T}}_V \right|, \\
\|\gamma_1 + \gamma_2\|^{\mathcal{T}_n}_V &= \|\gamma_1\|^{\mathcal{T}_n}_V + \|\gamma_2\|^{\mathcal{T}_n}_V.
\end{aligned}$$

We say a tree $\mathcal{T}$ satisfies, or is a model of, a formula ϕ, if and only if, the interpretation of ϕ under $\mathcal{T}$ and any valuation V is not empty, that is, $[\![\phi]\!]^{\mathcal{T}}_V \neq \emptyset$. A formula is valid, if and only if, it is satisfied by every tree. It is easy to see a formula ϕ is valid, if and only if, $\neg\phi$ is unsatisfiable.

The satisfiability problem for μTLIC consists in deciding whether or not a given formula is satisfiable.

We now give an intuition about the interpretation of formulas: propositions p are node labels; negation and disjunction are interpreted as the complement and the 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 siblings relation, respectively; $\mu x.\phi$ is interpreted as the least fixed-point; a counting formula $\gamma > b$ hold in a node, if and only if, the number of its children where γ true satisfy the corresponding constraint $> b$, respectively, for instance, $p_1 + (-2)p_2 > 0$ holds in nodes where the number of children where p_1 holds is greater than twice the number of children where p_2 is true.

We also use the following traditional notation:

$$\top := p \vee \neg p, \qquad \bot := \neg\top,$$
$$\phi \wedge \psi := \neg(\neg\phi \vee \neg\psi), \qquad [m]\,\phi := \neg\langle m\rangle\neg\phi,$$
$$\gamma \leq b := \neg(\gamma \leq b).$$

Note that $\top$ is true in every node, hence $\bot$ in none, 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, and $\gamma \leq b$ is true in nodes where the corresponding children satisfy $\leq b$. Other common counting operators can also be expressed, for instance, $\gamma = b$ can be written instead of $(\gamma \leq b) \wedge (\gamma > b - 1)$, where $b > 0$. Below in further sections, we also write $b_1 \# \gamma \# b_2$ ($\# \in \{>, \leq, =\}$) instead of $(\gamma \# b_1) \wedge (\gamma \# b_2)$.

Examples

Consider for instance the following formula ψ:

$$p \wedge (r \leq 2q),$$

where $(r \leq 2q)$ stands for $1r + (-2)q \leq 0$. ψ is true in nodes labeled by p, such that the number of its q children is at least twice the number of its r children. This is a common example that goes beyond the expressive power of (graded) μ-calculus and regular languages [16].

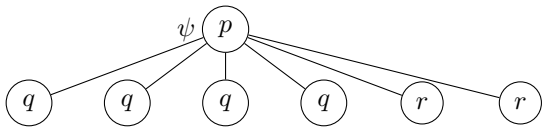


Fig. 2. A model for $\psi := (p \wedge [r \leq 2q])$

In Figure 2, there is a graphical representation of a model for ψ:

It is also possible to express recursive navigation, for example, consider the following formula ϕ:

$$\mu x.\psi \vee \langle\downarrow\rangle x,$$

ϕ is true in nodes with at least one descendant where ψ is true, that is, ϕ recursively navigates along children until it finds a ψ node.

Backward navigation may also be expressible with the help of inverse programs (converse modalities). For instance, consider the following formula φ:

$$\mu x.\psi \vee \langle\uparrow\rangle x,$$

φ holds in nodes with an ancestor where ψ is true, that is, φ recursively navigates along parents until it finds a ψ node. Furthermore, in contrast with other approaches without converse modalities [16, 32], this new feature allows to count also on sibling nodes. For instance:

$$\langle\uparrow\rangle(p > 10)$$

holds in nodes with more than 10 siblings named p.

2.2 Other Forms of Counting

We now show how several notions of counting (nominals, graded modalities and global counting) can also be expressed in terms of μTLIC formulas.

Hybrid Logics

The interpretation of nominals is a singleton, that is, nominals are formulas which are true in exactly one node in the entire model [9]. Now, it is easy to see that μTLIC can navigate recursively thanks to the fixed-points, and in all directions thanks to inverse programs.

Hence, μTLIC can then express for a formula to be true in one node while being false in all other

nodes of the model. Nominals are then defined as follows:

$$\begin{aligned} nom(\phi) =& \phi \wedge Siblings(\neg\phi \wedge Descendants(\neg\phi)), \\ & \wedge Ancestors(\neg\phi \wedge Siblings(\neg\phi)), \\ & \wedge Descendants(\neg\phi), \end{aligned}$$

where formulas $Siblings(\phi)$, $Ancestors(\phi)$, and $Descendants(\phi)$ are true, if and only if, ϕ is true in all siblings, ancestors, and descendants, respectively. More precisely:

$$\begin{aligned} Nav_m(\phi) :=& \langle m\rangle \mu x.\phi \wedge (\langle m\rangle x \vee \neg\langle m\rangle \top), \\ Siblings(\phi) :=& Nav_{\rightarrow}(\phi) \wedge Nav_{\leftarrow}(\phi), \\ Descendants(\phi) :=& Nav_{\downarrow}(\phi \wedge Siblings(\phi)), \\ Ancestors(\phi) :=& Nav_{\uparrow}(\phi). \end{aligned}$$

Graded Logics

In modal logics, graded modalities are specialized operators for expressing numerical bounds on the occurrence of a sole formula in adjacent nodes. In the context of tree models, the numerical bounds are on children nodes [4]. For instance, formula $\langle\downarrow, k\rangle\phi$ holds in nodes with *at least* $k+1$ children where ϕ is true. More precisely, given a tree structure $\mathcal{T}$ and a valuation V, graded formulas are interpreted as follows:

$$\begin{aligned} [\![\langle\downarrow, k\rangle\phi]\!]^{\mathcal{T}}_V &= \left\{ n \mid \left| \left\{ n' \mid n' \in \mathcal{R}(n,\downarrow) \cap [\![\phi]\!]^{\mathcal{T}}_V \right\} \right| > k \right\}, \\ [\![[\downarrow, k]\,\phi]\!]^{\mathcal{T}}_V &= [\![\neg\langle\downarrow, k\rangle\neg\phi]\!]^{\mathcal{T}}_V, \end{aligned}$$

where k is a positive integer number encoded in binary. $[\downarrow, k]\,\phi$ formulas are true in nodes with *all but at most* k children satisfying ϕ. It is then easy to see that Presburger formulas can express graded modalities, more precisely, for any tree $\mathcal{T}$ and valuation V, we have that:

$$[\![\phi > k]\!]^{\mathcal{T}}_V = [\![\langle\downarrow, k\rangle\phi]\!]^{\mathcal{T}}_V.$$

Global Counting

Global numerical constraints, as its name suggest, are operators used to impose constraints on the occurrence of a sole formula with respect to a constant in the entire model [37, 2, 39, 5].

That is, a formula $\phi >_G k$ holds in the entire model, if and only if, ϕ is satisfied by at least $k+1$ nodes. More precisely, the interpretation of global counting formulas with respect to a tree $\mathcal{T}$ and a valuation V is the following:

$$[\![\phi >_G k]\!]^{\mathcal{T}}_V = \begin{cases} \mathcal{N} & \text{if } \left| [\![\phi]\!]^{\mathcal{T}}_V \right| > k, \\ \emptyset & \text{otherwise.} \end{cases}$$

Note that the intended interpretation of $\phi \leq_G k$ is the same as for formula $\neg(\phi >_G k)$. In [5], it was shown that regular path queries (XPath) with numerical constraints on any path, for instance ancestors or descendants, can be succinctly expressed by global counting formulas. It was also shown in [5] that global numerical constraints does not provided extra expressive power by means of a reduction to the two-way μ-calculus (without counting). More precisely, for any binary tree T (for technical convenience, binary trees are used instead of n-ary trees, there is a well known bijection between them, see Figure 3 on page 7) and valuation V, we have the following:

$$[\![\phi >_G k]\!]^{T}_V = [\![\mu x.(C^{\phi}_k \wedge r) \vee \langle\uparrow\rangle x \vee \langle\leftarrow\rangle x]\!]^{T}_V,$$

where r stands for the root node $\neg\langle\uparrow\rangle\top \wedge \neg\langle\leftarrow\rangle\top$, and C^{ϕ}_k counts at least $k+1$ occurrences of ϕ in descendant nodes. More precisely:

$$\begin{aligned} C^{\phi}_0 :=& \mu x.\phi \vee \langle\downarrow\rangle x \vee \langle\rightarrow\rangle x, \\ C^{\phi}_1 :=& \mu x.\left(\phi \wedge \left(\langle\downarrow\rangle C^{\phi}_0 \vee \langle\rightarrow\rangle C^{\phi}_0\right)\right) \vee, \\ & \left(\neg\phi \wedge \langle\downarrow\rangle C^{\phi}_0 \wedge \langle\rightarrow\rangle C^{\phi}_0\right) \vee \langle\downarrow\rangle x \vee \langle\rightarrow\rangle x, \\ C^{\phi}_i :=& \mu x.\left(\phi \wedge \left(\langle\downarrow\rangle C^{\phi}_{i-1} \vee \langle\rightarrow\rangle C^{\phi}_{i-1}\ , \right.\right. \\ & \left.\left. \vee \bigvee_{k_1+k_2=i-2} \langle\downarrow\rangle C^{\phi}_{k_1} \wedge \langle\rightarrow\rangle C^{\phi}_{k_2}\right)\right), \\ & \vee \left(\neg\phi \wedge \bigvee_{k_1+k_2=i-1} \langle\downarrow\rangle C^{\phi}_{k_1} \wedge \langle\rightarrow\rangle C^{\phi}_{k_2}\right), \\ & \vee \langle\downarrow\rangle x \vee \langle\rightarrow\rangle x. \end{aligned}$$

μTLIC can therefore also express global numerical constraints. However, it is not hard to see that hardcoding of global numerical constraints comes at an exponential cost [5].

3 Satisfiability

In the current section, we describe a satisfiability algorithm for the logic μTLIC. That is, given an input μTLIC formula, the algorithm decides whether or not there is a tree model satisfying the formula. The algorithm inspired from the well-known Fischer-Ladner approach [18] and the binary encoding of counting constraints introduced in [5, 4]. Candidate trees are enumerated starting from the single nodes (leaves). Then, parents are iteratively added until a satisfying tree is found. The stop condition for this iterative process is given by the number of available nodes, which are defined as sets of subformulas (of the input formula). These subformulas represents the information required to build the trees: node names, tree topology, Presburger constraints. One notable distinction of our algorithm is that Presburger constraints are encoded in binary form.

Before describing the algorithm, we describe the notion of trees. Then we show that the algorithm is correct and in EXPTIME.

3.1 Fischer-Ladner-Presburger Trees

We first give a detailed description of the syntactic version of tree models constructed by the satisfiability algorithm.

Some preliminaries are now introduced. There is well-known bijection between binary and n-ary trees [22]. One adjacency is interpreted as the first child relation and the other adjacency is for the right sibling relation. In Figure 3 is depicted an example of the bijection. Hence, without loss of generality, from now on, we consider binary unranked trees only. At the logic level, formulas are interpreted as expected: $\langle\downarrow\rangle\phi$ holds in nodes such that ϕ is true in its first child; $\langle\rightarrow\rangle\phi$ holds in nodes where ϕ is satisfied by its right (following) sibling; $\langle\uparrow\rangle\phi$ is true in nodes whose parent satisfies ϕ; and $\langle\leftarrow\rangle\phi$ satisfies nodes where ϕ holds in its left (previous) sibling.

For the satisfiability algorithm we consider formulas in negation normal form only.

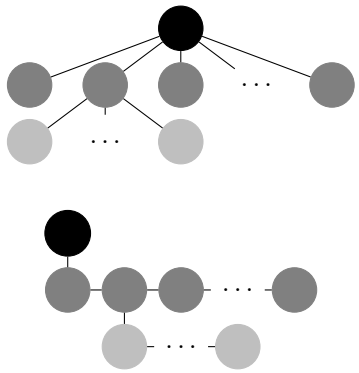


Fig. 3. Bijection of n-ary and binary trees

The negation normal form (NNF) of μTLIC formulas is defined by the usual De Morgan rules and the following ones:

$$
\begin{aligned}
\mathsf{nnf}(\neg p) &:= \neg p,\\
\mathsf{nnf}(\neg x) &:= \neg x,\\
\mathsf{nnf}(\neg(\phi \vee \psi)) &:= \mathsf{nnf}(\neg\phi) \wedge \mathsf{nnf}(\neg\psi),\\
\mathsf{nnf}(\neg(\phi \wedge \psi)) &:= \mathsf{nnf}(\neg\phi) \vee \mathsf{nnf}(\neg\psi),\\
\mathsf{nnf}(\neg\langle m\rangle\phi) &:= \langle m\rangle\mathsf{nnf}(\neg\phi) \vee \neg\langle m\rangle\top,\\
\mathsf{nnf}(\neg\mu x.\phi) &:= \mu x.\mathsf{nnf}(\neg\phi)\,[^{x}/_{\neg x}]\,,\\
\mathsf{nnf}(\neg(\gamma > b)) &:= \gamma \leq b,\\
\mathsf{nnf}(\neg(\gamma \leq b)) &:= \gamma > b.
\end{aligned}
$$

Hence, negation symbol $\neg$ in formulas in NNF occurs only in front of propositions and formulas of the form $\langle m\rangle\top$. It is also easy to see that the negation normal form of a formula has linear size with respect to the size of the formula. Also notice that we consider an extension of μTLIC formulas consisting of conjunctions, $\gamma \leq b$, and $\top$ formulas, with the expected semantics.

From now on, we often write $\gamma\#b$ to denote any of the following formulas: $\gamma > b$ or $\gamma \leq b$.

We now consider a binary encoding of natural numbers. Given a finite set of propositions, the binary encoding of a natural number is the Boolean combination of propositions satisfying the binary representation of the given number. For example, number 0 is written $\bigwedge_{i\geq 0}\neg p_i$, and number 7 is $p_2 \wedge p_1 \wedge p_0 \wedge \bigwedge_{i>2}\neg p_i$ (111 in binary). The binary encoding of numbers is required in the definition

ISSN 2007-9737

of counters, which are used in the satisfiability algorithm to verify counting subformulas.

Definition 3 (Counters)**.** Given a formula ϕ and a number $b > 0$, a counter of ϕ set to k is defined by:

$$(C(\phi) = b) := \overline{\phi^i},$$

where $i \in \{0, \ldots, \lceil \log(b) \rceil\}$, $\overline{\phi^i}$ is a sequence of propositions ϕ^i occurring positively in the binary encoding of b.

Consider for instance the counter of formula ϕ set to 7:

$$(C(\phi) = 7) := \phi^0, \phi^1, \phi^2.$$

We write $(C(\phi) = b) \in S$, when each $\phi^i \in S$, where $(C(\phi) = b) := \overline{\phi^i}$.

A formula ϕ induces a set of counters corresponding to its counting subformulas. The bound on the number of propositions used by counters is given by $K(\phi)$, and it is proved in Theorem 5.

Nodes in Fischer-Ladner-Presburger trees are defined as sets of subformulas. These subformulas are extracted with the help of the Fischer-Ladner-Presburger Closure. Before defining the Closure, we define the following set of subformulas of a counting expression:

$$S(a\phi) = \{\phi\}, \quad S(\gamma_1 + \gamma_2) = S(\gamma_1) \cup S(\gamma_2).$$

We often write ϕ_γ to denote $\phi \in S(\gamma)$.

Now, consider the following binary relation R^{FLP} on the set of μTLIC formulas, for $i = 1, 2$, $\circ = \vee, \wedge$, $j = 0, \ldots, \lceil \log(K(\phi)) \rceil$ and each $\phi_\gamma \in S(\gamma)$:

$$\begin{aligned}
&R^{\mathsf{FLP}}\left(\psi, \mathsf{nnf}(\neg\psi)\right),\\
&R^{\mathsf{FLP}}\left(\psi_1 \circ \psi_2, \psi_i\right),\\
&R^{\mathsf{FLP}}\left(\langle m\rangle\psi, \psi\right),\\
&R^{\mathsf{FLP}}\left(\mu x.\psi, \psi\left[{}^{\mu x.\psi}/_{x}\right]\right),\\
&R^{\mathsf{FLP}}\left(\gamma\#b, \langle\downarrow\rangle\mu x.\phi_\gamma \vee \langle\rightarrow\rangle x\right),\\
&R^{\mathsf{FLP}}\left(\gamma\#b, \phi^j_\gamma\right),\\
&R^{\mathsf{FLP}}\left(\neg\psi, \psi\right).
\end{aligned}$$

Definition 4 (Fischer-Ladner-Presburger Closure)**.** Given a formula ϕ, the Fischer-Ladner-Presburger Closure of ϕ is defined as $\mathsf{CL}^{\mathsf{FLP}}(\phi) = \mathsf{CL}^{\mathsf{FLP}}_k(\phi)$, such that k is the smallest positive integer satisfying $\mathsf{CL}^{\mathsf{FLP}}_k(\phi) = \mathsf{CL}^{\mathsf{FLP}}_{k+1}(\phi)$, where for $i \geq 0$:

$$\begin{aligned}
\mathsf{CL}^{\mathsf{FLP}}_0(\phi) =& \{\phi\},\\
\mathsf{CL}^{\mathsf{FLP}}_{i+1}(\phi) =& \mathsf{CL}^{\mathsf{FLP}}_i(\phi),\\
&\cup \left\{\psi \mid R^{\mathsf{FLP}}(\psi', \psi), \psi' \in \mathsf{CL}^{\mathsf{FLP}}_i(\phi)\right\}.
\end{aligned}$$

Example 1. Consider the following formula:

$$\phi := p \wedge [(q - r) > 1] \wedge r > 0.$$

This formula holds in p nodes with at least one more q child with respect to r children. In Figure 4, there is graphical representation of a ϕ-tree (Definition 7) for formula ϕ. In the definition of ϕ-trees, we use the notion of Fischer-Ladner-Presburger closure, which in the case of ϕ is defined as follows for $j = 0, 1, 2$:

$$\begin{aligned}
\mathsf{CL}^{\mathsf{FLP}}(\phi) =& \{p \wedge [(q - r) > 1] \wedge (r > 0)),\\
&p \wedge [(q - r) > 1], (r > 0), p, [(q - r > 1)],\\
&q, r, q^j, r^j, \langle\downarrow\rangle\mu x.q \vee \langle\rightarrow\rangle x, \langle\downarrow\rangle\mu x.r \vee \langle\rightarrow\rangle x\},\\
&\cup \mathsf{CL}^{\mathsf{FLP}}(\mathsf{nnf}(\phi)).
\end{aligned}$$

We are now ready to define the lean set for nodes in Fischer-Ladner-Presburger trees. The lean set contains the propositions, modal subformulas, counters and counting subformulas of the formula in question (for the satisfiability algorithm). Intuitively, propositions will serve to label nodes, modal subformulas contain the topological information of the trees, and counters are used to verify the satisfaction of counting constraints.

Definition 5 (Lean)**.** Given a formula ϕ, its lean set is defined as follows:

$$\begin{aligned}
\mathsf{lean}(\phi) = &\left\{p, \langle m\rangle\phi, \gamma\#b, \psi^i_\gamma \in \mathsf{CL}^{\mathsf{FLP}}(\phi)\right\}\\
&\cup \{\langle m\rangle\top, p'\},
\end{aligned}$$

provided that p' does not occur in ϕ, and $m = \downarrow, \rightarrow, \uparrow, \leftarrow$.

Example 2. Consider again the formula $\phi := p \wedge [(q-r) > 1] \wedge r > 0$ of Example 1, then for $i = 1, 2$ and $j = 0, 1, 2$, we have that:

$$\begin{aligned}\mathsf{lean}(\phi) =&\{p, q, r, q^j, r^j, \langle\downarrow\rangle\psi_i, \langle\rightarrow\rangle\psi_i, \langle\downarrow\rangle\mathsf{nnf}(\psi_j),\\ &\langle\rightarrow\rangle\mathsf{nnf}(\psi_j), (q-r)\#1, r\#0, \langle m\rangle\top, p'\},\end{aligned}$$

where
$\psi_1 = \mu x.q \vee \langle\rightarrow\rangle x$ and $\psi_2 = \mu x.r \vee \langle\rightarrow\rangle x$.

Recall that q^j and r^j are the corresponding propositions (associated to q and r) required to define the corresponding binary encoding of numbers.

Definition 6. The set of ϕ-nodes is defined as follows:

$$\begin{aligned}\mathcal{N}^\phi =&\{n^\phi \subseteq \mathsf{lean}(\phi) \mid p \in n^\phi,\\ &\langle m\rangle\psi \in n^\phi \Rightarrow \langle m\rangle\top \in n^\phi,\\ &\langle\uparrow\rangle\top \in n^\phi \Leftrightarrow \langle\leftarrow\rangle\top \notin n^\phi\}.\end{aligned}$$

Intuitively, a ϕ-node n^ϕ is defined as a subset of the lean, such that:

— at least (exactly[1]) one proposition (different from the counter propositions) occurs in n^ϕ;

— if a modal subformula $\langle m\rangle\psi$ occurs in n^ϕ, then $\langle m\rangle\top$ also does; and

— both $\langle\uparrow\rangle\top$ and $\langle\leftarrow\rangle\top$ can not occur in n^ϕ.

When it is clear from the context, ϕ-nodes are called simply nodes.

We are finally ready to define the Fischer-Lander-Presburger ϕ-trees.

Definition 7. A ϕ-tree is defined:

— either as empty $\emptyset$, or

— as a triple $(n^\phi, T_1^\phi, T_2^\phi)$, provided that n^ϕ is a ϕ-node and T_i^ϕ $(i = 1, 2)$ are ϕ-trees.

The root of $(n^\phi, T_1^\phi, T_2^\phi)$ is n^ϕ. We often call ϕ-trees simply trees.

[1] In the XML setting, exactly one proposition occurs at each node.

Example 3. Consider again the following formula $\phi := p \wedge [(q-r) > 1] \wedge r > 0$. Then $T = (n_0, (n_1, \emptyset, (n_2, \emptyset, (n_3, \emptyset, (n_4, \emptyset, \emptyset)))), \emptyset)$ is a ϕ-tree, where:

$$\begin{aligned}n_0 =&\{p, C(q) = 0, C(r) = 0, (q-r) > 1, r > 0,\\ &\langle\downarrow\rangle\psi_1, \langle\downarrow\rangle\psi_2, \langle\downarrow\rangle\top\},\\ n_1 =&\{q, C(q) = 3, C(r) = 1, (q-r) \leq 1, r \leq 0,\\ &\langle\rightarrow\rangle\psi_1, \langle\rightarrow\rangle\psi_2, \langle\uparrow\rangle\top, \langle\rightarrow\rangle\top\},\\ n_2 =&\{q, C(q) = 2, C(r) = 1, (q-r) \leq 1, r \leq 0,\\ &\langle\rightarrow\rangle\psi_1, \langle\rightarrow\rangle\psi_2, \langle\leftarrow\rangle\top, \langle\rightarrow\rangle\top\},\\ n_3 =&\{q, C(q) = 1, C(r) = 1, (q-r) \leq 1, r \leq 0,\\ &\langle\rightarrow\rangle\psi_2, \langle\leftarrow\rangle\top, \langle\rightarrow\rangle\top\},\\ n_4 =&\{r, C(q) = 0, C(r) = 1, (q-r) \leq 1, r \leq 0,\\ &\langle\leftarrow\rangle\top\}.\end{aligned}$$

ϕ-nodes n_i $(i = 0, \ldots, 4)$ are defined from the lean of ϕ (Example 2). In Figure 4 is depicted a graphical representation of T. Notice that counters in the root n_0 are set to zero 0, that is, no proposition corresponding to counters occurs. This is because counters are intended to count siblings only. For instance, counters in n_1 are set to 3 and 1 for q and r, respectively, because there are three q's and one r in n_1 and its siblings. Counting formulas occur positively only at the root n_0, because they are intended to be true when the counters in the children of n_0 satisfy the Presburger constraints. Since n_i $(i > 0)$ does not have children, then counting formulas occur negatively (recall the negation normal form of the input formula is also in the lean) in these nodes. Finally, notice that modal subformulas define the topology of the tree.

3.2 The Algorithm

We now define a satisfiability algorithm for the logic μTLIC following the Fischer-Ladner method [5, 4, 16, 18]. Given an input formula, the algorithm decides whether or not the formula is satisfiable. The algorithm builds ϕ-trees in a bottom-up manner. Starting from the leaves, parents are iteratively added until a satisfying tree, with respect to ϕ, is found.

Algorithm 1 describes the bottom-up construction of ϕ-trees.

ISSN 2007-9737

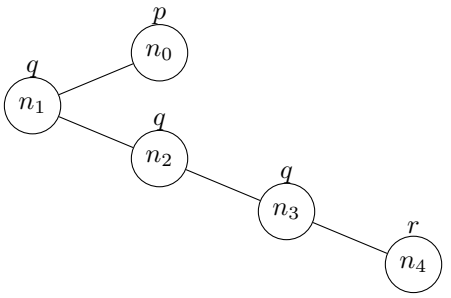


Fig. 4. ϕ-tree model for $\phi = p \wedge [(q - r) > 1] \wedge (r > 0)$ built by the satisfiability algorithm in 5 steps

Algorithm 1 Satisfiability algorithm for μTLIC

$Y \leftarrow \mathcal{N}^{\phi}$
$X \leftarrow Init(\phi)$
$X' \leftarrow \emptyset$
while $X \nvdash \phi$ and $X \neq X'$ **do**
  $X' \leftarrow X$
  $X \leftarrow Update(X', Y)$
  $Y \leftarrow Y \setminus root(X)$
end while
if $X \vdash \phi$ **then**
  return ϕ is satisfiable
end if
return ϕ is not satisfiable

The set $Init(\phi)$ gathers the leaves. The satisfiability of formulas with respect to ϕ-trees is tested with the entailment relation $\vdash$. Inside the loop, the $Update$ function consistently adds parents to previously build trees until either a satisfying tree is found or no more trees can be built. If a satisfying tree is found, the algorithm returns the input formula is satisfiable, otherwise, the algorithm returns the input formula is not satisfiable.

Example 4. Consider the formula $\phi := p \wedge [(q-r) > 1] \wedge (r > 0)$. The ϕ-tree T, described in Example 3, is built by the satisfiability algorithm in 5 steps. All leaves are first defined by $Init(\phi)$ (Definition 9): notice that n_4 is a leaf because it does not contain downward modal formulas. Once in the *while* cycle, parents and previous siblings are iteratively added to previously built trees, which by the second step consists of leaves only: since $\langle\downarrow\rangle(\mu x.r \vee \langle\rightarrow\rangle x)$ and $\langle\downarrow\rangle\top$ occur in n_3, and r and $\langle\uparrow\rangle\top$ occur in n_4, it is clear n_3 can be the parent of n_4, analogously for n_2 and n_3, and n_1 and n_2, respectively; also it is clear that n_0 can be a parent of n_1. Notice that n_0 is the root due to the absence of upward modal formulas $\langle\uparrow\rangle\top$ and $\langle\leftarrow\rangle\top$. The construction of T is graphically represented in Figure 4.

We now give a detailed description of the algorithm components.

Definition 8 (Entailment)**.** The entailment relation is defined as follows:

$$\frac{}{n \vdash \top} \qquad \frac{\phi \in n}{n \vdash \phi} \qquad \frac{\phi \notin n}{n \vdash \neg\phi}$$

$$\frac{n \vdash \phi \quad n \vdash \psi}{n \vdash \phi \wedge \psi} \qquad \frac{n \vdash \phi}{n \vdash \phi \vee \psi} \qquad \frac{n \vdash \psi}{n \vdash \phi \vee \psi}$$

$$\frac{n \vdash \phi\left[{}^{\mu x.\phi}/_{x}\right]}{n \vdash \mu x.\phi}.$$

If there is a node n in a tree T, such that n entails ϕ ($n \vdash \phi$) and formulas $\langle\uparrow\rangle\top$ and $\langle\leftarrow\rangle\top$ does not occur in the root of T, we then say that the tree T entails ϕ, $T \vdash \phi$. Given a set of trees X, if there is a tree T in X entailing ϕ ($T \vdash \phi$), then X entails ϕ, $X \vdash \phi$. Relation $\nvdash$ is defined in the obvious manner.

Leaves are ϕ-nodes without downward adjacencies, that is, formulas with the form $\langle\downarrow\rangle\psi$ or $\langle\rightarrow\rangle\psi$ do not occur in leaves. Also, counters are properly initialized, that is, for each counting subformula $\gamma\#b$ of the input formula, if a leaf satisfies ϕ_γ, then $C(\phi_\gamma) = 1$ is contained in the leaf, otherwise $C(\phi_\gamma) = 0$, that is, no counting proposition corresponding to ϕ_γ occurs in the leaf. The set of leaves is defined by the $Init$ function.

Definition 9 (Init)**.** Given a formula ϕ, its initial set $Init(\phi)$ is defined as follows:

$$\{n^{\phi} \in \mathcal{N}^{\phi} \mid \langle\downarrow\rangle\top, \langle\rightarrow\rangle\top \notin n^{\phi}$$
$$\gamma\#b \in \mathsf{lean}(\phi), \phi_\gamma \in n^{\phi} \Rightarrow [C(\phi_\gamma) = 1] \in n^{\phi}$$
$$\gamma\#b \in \mathsf{lean}(\phi), \phi_\gamma \notin n^{\phi} \Rightarrow [C(\phi_\gamma) = 0] \in n^{\phi}\}.$$

Notice that, from definition of ϕ-nodes, if formulas of the forms $\langle\downarrow\rangle\top$ and $\langle\rightarrow\rangle\top$ do not occur in leaves, then neither formulas of the forms $\langle\downarrow\rangle\psi$ and $\langle\rightarrow\rangle\psi$ do.

Example 5. Consider again the formula ϕ of Example 3. It is then easy to see that n_4 is a leaf. n_4 does not contain downward modal formulas $\langle\downarrow\rangle\psi$ and $\langle\rightarrow\rangle\psi$. Also, counters are properly initialized in n_4, i.e., $C(r) = 1$ occurs in n_4.

The $Update$ function consistently adds parents to previously built trees. Consistency is defined with respect to two different notions. One notion is with respect to modal formulas. For example, a modal formula $\langle\downarrow\rangle\phi$ is contained in the root of a tree, if and only if, its first child satisfies ϕ.

Definition 10 (Modal Consistency)**.** Given a ϕ-node n^ϕ and a ϕ-tree T with root r, n^ϕ and T are m modally consistent $\Delta_m(n^\phi, T)$, if and only if, for all $\langle m\rangle\psi, \langle\overline{m}\rangle\phi$ in lean(ϕ), where $m \in \{\downarrow, \rightarrow\}$ and $\overline{m} \in \{\uparrow, \leftarrow\}$, we have that:

$$\langle m\rangle\psi \in n^\phi \Leftrightarrow r \vdash \psi,$$
$$\langle\overline{m}\rangle\psi \in r \Leftrightarrow n^\phi \vdash \psi.$$

Example 6. Consider ϕ in Figure 4. In step 2, it is easy to see that n_3 is modally consistent with n_4: formula $\langle\rightarrow\rangle\mu x.r \vee \langle\rightarrow\rangle x$ is clearly true in n_3, because r occurs in n_4. In the following steps, n_i is clearly modally consistent with n_{i+1}.

Another consistency notion is defined in terms of counters. Since the first child is the upper one in a tree, it must contain all the information regarding counters, i.e., each time a previous sibling is added by the algorithm, counters must be updated. Counter consistency must also consider that counting formulas occurs in the parents, if and only if, the counters of its first child are consistent with constraints in counting subformulas.

Definition 11 (Counter Consistency)**.** Given a ϕ-node n^ϕ and trees T_1 and T_2, we say that n^ϕ and T_1 and T_2 are counter consistent, written $\Theta(n^\phi, T_1, T_2)$, if and only if, for the roots r_1 and r_2 of T_1 and T_2, respectively, and for all counting formulas $\gamma\#b$ in lean(ϕ), we have that:

$$(C(\phi_\gamma) = b') \in n^\phi, n^\phi \vdash \phi_\gamma \Leftrightarrow (C(\phi_\gamma) = b' - 1) \in r_2,$$
$$(C(\phi_\gamma) = b') \in n^\phi, n^\phi \nvdash \phi_\gamma \Leftrightarrow (C(\phi_\gamma) = b') \in r_2,$$
$$(\gamma\#b) \in n^\phi \Leftrightarrow \forall\,(C(\phi_\gamma) = b') \in r_1 : \gamma\left[{}^{b'}/_{\phi_\gamma}\right]\#b.$$

Example 7. Consider the formula ϕ of Example 3 and Figure 4. In steps 2, 3 and 4, since previous siblings are added, counters for q are incremented in n_3, n_2 and n_1, respectively. In step 5, the counting formulas $q - r > 1$ and $r > 0$ are present in the root n_0, due to the fact that counters, in the first child, satisfy the Presburger constraints.

$Update$ function gathers the notions of counter and modal consistency.

Definition 12 (Update)**.** Given a set of ϕ-trees X and set of ϕ-nodes Y, the update function is defined as follow for $i = 1, 2$:

$$Update(X, Y) = \{(n^\phi, T_1, T_2) \mid T_i \in X, n^\phi \in Y, \Delta_i(n^\phi, T_i), \Theta(n^\phi, T_1, T_2)\}.$$

We finally define the function $root(X)$, which takes as input a set of ϕ-trees and returns a set with the roots of the ϕ-trees.

3.3 Correctness

We now show that Algorithm 1 is correct, and then we describe its complexity. Correctness is shown by proving that the algorithm is sound and complete. For these proofs, we first need a fixed point theorem. Proving substitution is monotone is the first step.

Lemma 1. *Given a μTLIC formula ϕ, a tree structure $\mathcal{T} = (P, \mathcal{N}, \mathcal{R}, L)$ and a valuation V, let $f : \mathcal{P}(\mathcal{N}) \mapsto \mathcal{P}(\mathcal{N})$ be defined as $f(S) = [\![\phi]\!]^{\mathcal{T}}_{V[S/x]}$, where x is a free variable in ϕ. If $S \subseteq S'$, then $f(S) \subseteq f(S')$.*

Proof. We proceed by induction on the structure of ϕ. Base cases are trivial and most inductive cases are straightforward by inductive hypothesis. Recall that since we are considering only formulas in negated normal form, negation symbols $\neg$ occur in front of propositions and formulas $\langle m\rangle\top$ only. Consider for instance the case of conjunction $\psi \wedge \varphi$. By inductive hypothesis we know $[\![\psi]\!]^{\mathcal{T}}_{V[S/x]} \subseteq [\![\psi]\!]^{\mathcal{T}}_{V[S'/x]}$ and $[\![\varphi]\!]^{\mathcal{T}}_{V[S/x]} \subseteq [\![\varphi]\!]^{\mathcal{T}}_{V[S'/x]}$, hence $[\![\psi]\!]^{\mathcal{T}}_{V[S/x]} \cap [\![\varphi]\!]^{\mathcal{T}}_{V[S/x]} \subseteq [\![\psi]\!]^{\mathcal{T}}_{V[S'/x]} \cap [\![\varphi]\!]^{\mathcal{T}}_{V[S'/x]}$, and then $[\![\psi \wedge \psi]\!]^{\mathcal{T}}_{V[S/x]} \subseteq [\![\psi \wedge \varphi]\!]^{\mathcal{T}}_{V[S'/x]}$.

ISSN 2007-9737

The case of the Presburger formula $\gamma > b$ is more interesting. We prove this case by a second induction on the structure of γ. We distinguish three base cases, the first one is $a\psi > b$. Notice in this case $a \geq 0$. By the first inductive hypothesis we know $\left|[\![\psi]\!]^{\mathcal{T}}_{V[^{S}/_{x}]}\right| \leq \left|[\![\psi]\!]^{\mathcal{T}}_{V[^{S}/_{x}]}\right|$. It is then clear that, for any node n, $\|\psi\|^{\mathcal{T}_n}_{V[^{S}/_{x}]} \leq \|\psi\|^{\mathcal{T}_n}_{V\left[^{S'}/_{x}\right]}$. The second base case is $a_1\gamma_1 + a_2\gamma_2 > b$, where both a_1 and a_2 are non-negative integers. This is straightforward from the first base case. The third base case is $a_1\gamma_1 - a_2\gamma_2 > b$. From the first base case, it is easy to see that for any n and $i = 1, 2$, $\|a_i\gamma_i\|^{\mathcal{T}_n}_{V[^{S}/_{x}]} \leq \|a_i\gamma_i\|^{\mathcal{T}_n}_{V\left[^{S'}/_{x}\right]}$. Hence, $\|a_1\gamma_1\|^{\mathcal{T}_n}_{V[^{S}/_{x}]} - \|a_2\gamma_2\|^{\mathcal{T}_n}_{V[^{S}/_{x}]} \leq \|a_1\gamma_1\|^{\mathcal{T}_n}_{V\left[^{S'}/_{x}\right]} - \|a_2\gamma_2\|^{\mathcal{T}_n}_{V\left[^{S'}/_{x}\right]}$. The inductive step in $\gamma > b$ is immediate from the bases cases. For the case $\gamma \leq b$, recall that in order to ensure the fixed-point existence, variables can only occur positively, hence, variables in γ occur negatively. We then proceed analogously as in the case $\gamma > b$.

Consider now the case for $\mu y.\psi$. Now let $S_i \subseteq \mathcal{N}$ be defined by $g(S_i, S) \subseteq S_i$, where $g(S_i, S) = [\![\psi]\!]^{\mathcal{T}}_{V\left[^{S_i}/_{y}\right][^{S}/_{x}]}$, and $S'_i \subseteq \mathcal{N}$ by $g(S_i, S') \subseteq S'_i$. Now let $S_0 = \bigcap_{\forall i} S_i$ and $S'_0 = \bigcap_{\forall i} S'_i$. Note that $f(S) = S_0$ and $f(S') = S'_0$. By inductive hypothesis we know $g(S_i, S) \subseteq g(S_i, S')$ for every i. Now by transitivity of the subset relation (recall $g(S_i, S') \subseteq S'_i$) we obtain $g(S_0, S) \subseteq S'_0$, that is, there is an i such that $S'_0 = S_i$. By definition of S_0, it is easy to see $S_0 \subseteq S_i$ for every i. We then conclude $S_0 \subseteq S'_0$.

□

We now prove the fixed point Theorem.

Theorem 1. *Given a μTLIC formula ϕ, a tree structure $\mathcal{T} = (P, \mathcal{N}, \mathcal{R}, L)$ and a valuation V, let $f : \mathcal{P}(\mathcal{N}) \mapsto \mathcal{P}(\mathcal{N})$ be defined as $f(S) = [\![\phi]\!]^{\mathcal{T}}_{V[^{S}/_{x}]}$, where x is a free variable in ϕ, then the least fixed point of f is $\bigcap\{\mathcal{N}' \subseteq \mathcal{N} \mid f(\mathcal{N}') \subseteq \mathcal{N}'\}$.*

Proof. Since $\mathcal{N}$ is finite, then there is a finite number of $S_i \subseteq \mathcal{N}$, such that $f(S_i) \subseteq S_i$. Let $S = \bigcap_{\forall i} S_i$. Since $S \subseteq S_i$ for every i, and by Lemma 1, we obtain that $f(S) \subseteq f(S_i)$. By definition of each S_i, we also know $S \subseteq S_i$ for every i. Then by transitivity of the subset relation we obtain $f(S) \subseteq S_i$. Now recalling that $S = \bigcap_{\forall i} S_i$, $f(S) \subseteq S_i$ implies $f(S) \subseteq S$. Then by Lemma 1, $f(f(S)) \subseteq f(S)$. Then again by definition of S_i, there is a j such that $f(S) = S_j$. Since $S \subseteq S_j$, hence $S \subseteq f(S)$ and therefore $S = f(S)$. Now since $S \subseteq S_i$ for every i, it is then clear that S is the least fixed point. □

A straightforward observation from the fixed point Theorem 1 is that a fixed point $\mu x.\phi$ is equivalent to its unfolding $\phi\left[^{\mu x.\phi}/_{x}\right]$, that is, for any $\mathcal{T}$ and valuation V, we have that $[\![\mu x.\phi]\!]^{\mathcal{T}}_{V} = \left[\!\left[\phi\left[^{\mu x.\phi}/_{x}\right]\right]\!\right]^{\mathcal{T}}_{V}$.

Theorem 2 (Soundness)**.** *If the satisfiability algorithm returns that ϕ is satisfiable, then there is tree model satisfying ϕ.*

Proof. Assume T is the ϕ-tree that entails ϕ. Then we construct a tree model $\mathcal{T}$ isomorphic to T as follows:

— the nodes of $\mathcal{T}$ are the ϕ-nodes;

— for each triple (n, T_1, T_2) in T, $n_1 \in \mathcal{R}(n, \downarrow)$ and $n_2 \in \mathcal{R}(n, \rightarrow)$, provided that n_i are the roots of T_i $(i = 1, 2)$; and

— if $p \in n$, then $p \in L(n)$.

We now show by induction on the structure of the input formula ϕ that $\mathcal{T}$ satisfies ϕ.

Base cases are immediate, that is, when the input formula is either a proposition, a negated proposition or formulas with the form $\neg\langle m\rangle\top$.

Negations and disjunctions are also immediate by induction. Modal formulas $\langle m\rangle\phi$ are clearly satisfied by the construction of the model $\mathcal{T}$ and because ϕ is satisfied by induction. For counting formulas $a_1\psi_1 + a_2\psi_2 + \ldots + a_n\psi_n \# b$ recall by induction there is a node n in T such that $(C(\psi_1) = b_1), (C(\psi_2) = b_2), \ldots, (C(\psi_n) = b_n) \in n$ and $a_1b_1 + a_2b_2 + \ldots + a_nb_n \# b$.

Also in T, we know that n is the first child of a node n_0 where $a_1\psi_1 + a_2\psi_2 + \ldots + a_n\psi_n \# b$ is entailed. It is then easy to see that $\mathcal{T}$ satisfies $a_1\psi_1 + a_2\psi_2 + \ldots + a_n\psi_n \# b$ due to the construction described above. In the case of fixed-points, if $\mu x.\phi$ is entailed by T, then we know by the definition of the entailment relation

that $\phi\left[{}^{\mu x.\phi}/_{x}\right]$ is also entailed by T. In order to show that $\phi\left[{}^{\mu x.\phi}/_{x}\right]$ is satisfied by $\mathcal{T}$, we proceed by another structural induction on ϕ, which goes smothly since fixed-points are not considered (variables occur only in the scope of modalities or counting formulas). □

Completeness proof is divided in two main steps: first we show that there is a lean labeled version of the satisfying model; and then we show that the algorithm can actually build the lean labeled version of the tree model.

Theorem 3. *If there is a tree structure $\mathcal{T}$ satisfying a formula ϕ, then there is a Fischer-Ladner-Presburger ϕ-tree entailing ϕ.*

Proof. Assume $\mathcal{T}$ satisfies the formula ϕ. We construct a lean labeled version T of $\mathcal{T}$ as follows: the nodes and shape of T are the same as in $\mathcal{T}$; for each $\psi \in \text{lean}(\phi)$, if n in $\mathcal{T}$ satisfies ψ, then ψ is in n of T; and the counters are set in the nodes in T as the algorithm does in a bottom-up manner.

It is now shown by induction on the derivation of $T \vdash \phi$ that T entails ϕ. By the construction of T and by induction most cases are straightforward. For the fixed-point case $\mu x.\psi$, we proceed by induction on the structure of the unfolding $\psi\left[{}^{\mu x.\psi}/_{x}\right]$. That there is a finite unfolding comes from the Fixed-Point Theorem 1: $[\![\mu x.\phi]\!]_V^{\mathcal{T}} = [\![\phi\left[{}^{\mu x.\phi}/_{x}\right]]\!]_V^{\mathcal{T}}$. This induction is immediate because variables and hence unfolded fixed-points occur in the scope of modal or counting formulas only. □

Before proving that the algorithm builds T, we need to show that there are enough ϕ-nodes to construct T. Recall ϕ-nodes are lean subsets, and the lean is composed by propositions, modal and counting formulas occurring in the input formula, plus counters. Since counters count children nodes, we then need a bound on the number of children. For this purpose, we use a bound of an integer programming problem.

Theorem 4. *[30] Let A be a $m \times n$ integer matrix and b a m-vector, both with entries in $\mathbb{Z}$. Then if $Ax = b$ has a solution $x \in \mathbb{N}^n$, it also has one in $\{0, \ldots, n(ma)^{2m+1}\}^n$.*

Theorem 5. *If a formula ϕ is satisfiable, then there is a ϕ-tree entailing ϕ where each node has at most an exponential number of children with respect to the size of ϕ.*

Proof. Since ϕ is satisfiable, there is a ϕ-tree T entailing ϕ by Theorem 3. Recall each node in T is a subset of lean(ϕ), thus composed by propositions, modal formulas $\langle m \rangle \psi$ and counting formulas $\gamma \# b$. Notice formulas $\langle \downarrow \rangle \psi$ and $\gamma \# b$ are the ones enforcing child witnesses. Also notice $\langle \downarrow \rangle \psi$ are equivalent to $\psi > 0$. We now encode this set of counting formulas as an integer programming problem in order to obtain a bound on the number of required to children: $a_1\phi_1 + \ldots a_n\phi_n \# b$ as $a_1 x_{\phi_1} + \ldots + a_n x_{\phi_n} + x = b + 1$ when $\#$ is $>$, and $a_1 x_{\phi_1} + \ldots + a_n x_{\phi_n} - x = b$ when $\#$ is $\leq$, where $x \geq 0$. Then, each node has at most an exponential number of children by Theorem 4. □

We are now ready to show that the algorithm builds the lean labeled version T of the satisfying model $\mathcal{T}$.

Theorem 6 (Completeness)**.** *If there is a tree model $\mathcal{T}$ satisfying a formula ϕ, then the satisfiability algorithm returns that ϕ is satisfiable.*

Proof. The proof proceeds by induction on the height of $\mathcal{T}$. The base case is trivial. Consider now the induction step. By induction, we know that the left and right subtrees of T were built by the algorithm, we now show that the root n of T can be joined to the previously built left and right subtrees. This is true due to the following: $\Delta(n, n_i)$ is consistent with $\mathcal{R}$, where $i = 1, 2$ and n_i are the roots of the left and right subtrees, respectively; and by Theorem 5, there are at most an exponential number of children, with respect to the size of ϕ, distinguished by counters, encoded in binary by a linear number of propositions. □

Theorem 7 (Complexity)**.** *μTLIC satisfiability is EXPTIME-complete.*

Proof. We first show that the lean set of the input formula ϕ has linear size with respect to the size of ϕ. This is easily proven by induction on the structure ϕ and by Theorem 5: an exponential number of children can be distinguished by a linear

amount of counting propositions (recall counters are encoded in binary). We then proceed to show that the algorithm takes exponential time with respect to the size of the lean. Since ϕ-nodes are defined as subsets of the lean, it is then clear that the number of ϕ-nodes is single exponential with respect to lean size, then there is at most an exponential number of steps in the loop of the algorithm.

It remains to prove that each step, including the ones inside the loop, takes at most exponential time: computing $Init(\phi)$ implies the traversal of $\mathcal{N}^{\phi}$ and hences takes exponential time; testing $\vdash$ takes linear time with respect to the node size, and hence its cost is exponential with respect to the set of trees; and since the cost of relations of modal and counter consistency Δ_m and Θ is linear, then the $Update$ functions takes at most exponential time. Since the μ-calculus for trees is EXPTIME-complete [12], then μTLIC is hard for EXTPIME, therefore, also complete. □

4 Extended Regular Tree Languages

In this section, we introduce several extensions of regular tree languages, which encompass most XML schema languages used in practice, such as DTDs, XML Schema and RelaxNG [29, 22]. First we consider the extension with the interleaving operator [27]. Then the extension with counting operators [28]. We show that these extension can be linearly characterized by μTLIC. In Section 5, we also show that regular path queries (XPath) [36] with Presbuger constraints on children path can also be linearly expressed by μTLIC. As a consequence, μTLIC can be used as a framework for standard XML reasoning problems involving schemas and queries with counting and interleaving operators. In Section 3, we describe an EXPTIME satisfiability algorithm, which together with results described in this Section, imply new optimal (EXPTIME) bounds on emptiness, inclusion and equivalence of XPath queries (with counting) and XML schemas (with counting and interleaving).

4.1 Regular Tree Languages

We define the syntax of regular trees similarly as in [20, 22].

Definition 13 (Syntax of regular trees)**.** We define the set of regular tree expressions by the following grammar:

$$e := \epsilon \mid x \mid p[e] \mid e \cdot e \mid e + e \mid \text{let } \overline{x = e} \text{ in } e.$$

We write p instead of $p[\epsilon]$, and we consider $e \cdot \epsilon$ and $\epsilon \cdot e$ to be simply e.

Following [22], we now give a precise semantics of regular tree expressions, but first, we define the following notation. Consider a tree structure $\mathcal{T} = (P, \mathcal{N}, \mathcal{R}, L)$, recall P is a set of propositions, $\mathcal{N}$ a set of nodes, $\mathcal{R}$ is transition function among nodes forming a tree, and L is a function labeling nodes with propositions. Then, we write $(n, \overline{\mathcal{T}})$ to denote $\mathcal{T}$, with root n and children subtrees $\overline{\mathcal{T}} = \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$, that is $n_1, n_2, \ldots, n_k \in \mathcal{R}(n, \downarrow)$, where n_i is the root of $\mathcal{T}_i$ $(i = 1, \ldots, k)$. If we write $\overline{\mathcal{T}} \in S$, we mean the composition of $\mathcal{T}_i$ is in set S. By the composition of a sequence of trees $\overline{\mathcal{T}}$, written $\mathcal{T}_1 \circ \mathcal{T}_2 \ldots \circ \ldots \mathcal{T}_k$, we mean the resulting tree $(n, \overline{\mathcal{T}})$. The composition of two sets of trees S_1 and S_2, written $S_1 \circ S_2$, denotes the composition of all pairs of trees in S_1 and S_2, more precisely, the trees $\mathcal{T}_1 \circ \mathcal{T}_2$, such that $\mathcal{T}_i \in S_i$ $(i = 1, 2)$.

Definition 14 (Semantics of regular trees)**.** Given a valuation V (from variables to sets of trees), regular tree expressions are interpreted as follows:

$$\begin{aligned}
[\![\epsilon]\!]_V &= \{\emptyset\}, \\
[\![x]\!]_V &= V(x), \\
[\![p[e]]\!]_V &= \left\{ (n, \overline{\mathcal{T}}) \mid p \in L(n), \overline{\mathcal{T}} \in [\![e]\!]_V \right\}, \\
[\![e_1 \cdot e_2]\!]_V &= [\![e_1]\!]_V \circ [\![e_2]\!]_V, \\
[\![e_1 + e_2]\!]_V &= [\![e_1]\!]_V \cup [\![e_2]\!]_V, \\
[\![\text{let } \overline{x = e} \text{ in } e]\!]_V &= [\![e]\!]_{\text{lfp}\left(V\overline{\left[[\![e]\!]_V/_x\right]}\right)},
\end{aligned}$$

where $\text{lfp}\left(V\overline{\left[[\![e]\!]_V/_x\right]}\right)$ stands for the least fixed point of the substitution function.

Note that there is always a least fixed point due to the Knaster-Tarski Theorem [35] on fixed points (substitution is monotone with respect to the subset ordering).

Intuitively, regular tree expressions are interpreted as sets of unranked trees: ϵ is interpreted as the empty set; $p[e]$ denotes the sets of trees whose root is labeled by p and whose children are denoted by e; the interpretation of $e_1 \cdot e_2$ is the set of trees whose children are denoted by e_1 and e_2, from left to right; $e_1 + e_2$ is interpreted as the union of the interpretations of e_1 and e_2; and let $\overline{x = e}$ in e is interpreted as the least fixed point. The Kleene star operator can be expressed in terms of the least fixed point, for instance, the regular tree expression $p[q^\star]$ in Figure 1(a) can be written as follows:

$$\text{let } x = q \cdot (x + \epsilon) \text{ in } p[x + \epsilon].$$

We now show that regular tree expressions can linearly be translated in terms of the μ-calculus, as already shown in [4, 5]. This implies that traditional reasoning problems, such as emptiness, containment (inclusion) and equivalence, can be efficiently expressed in terms of the satisfiability of μ-calculus formulas. Before defining a translation function from regular tree expressions to logic formulas, we define $\mu\overline{x.\phi}.$ in ϕ as a generalization (several binded variables) of the fixed point operator with the expected semantics. This generalization does not provide more expressive power, although, it is more succinct [4].

Definition 15. We define the following translation function from regular tree expressions to μ-calculus formulas:

$$\begin{aligned}
F(\epsilon) &:= \bot,\\
F(x) &:= x,\\
F(p[e]) &:= p \wedge F^{\downarrow}(e),\\
F(e_1 \cdot e_2) &:= F(e_1) \wedge F^{\rightarrow}(e_2),\\
F(e_1 + e_2) &:= F(e_1) \vee F(e_2),\\
F(\text{let } \overline{x = e} \text{ in } e) &:= \mu\overline{x.F(e)}.\text{ in } F(e),
\end{aligned}$$

where $F^m(e)$ is defined as follows for $m \in \{\downarrow, \rightarrow\}$

— $\neg\langle m\rangle\top$ if e is ϵ,

— $\neg\langle m\rangle\top \vee F^m(e')$ if e has the forms $\epsilon + e'$, $e' + \epsilon$ and e' is nullable,

— $\neg\langle m\rangle\top \vee \langle m\rangle F(e')$ if e has the forms $\epsilon + e'$, $e' + \epsilon$ and e' is not nullable, and

— $\langle m\rangle F(e)$ otherwise.

We say an expression e is nullable when it is a variable bounded to an expression that can be interpreted as the empty tree, as for instance $\epsilon + e'$.

Now, consider as an example the expression $p[q^\star]$. This can be expressed in terms μTLIC as follows:

$$\begin{aligned}
F(p[q^\star]) :=& F(\text{let } x = q \cdot (x + \epsilon) \text{ in } p[x + \epsilon]),\\
:=& \mu x.q \wedge \neg\langle\downarrow\rangle\top \wedge (\neg\langle\rightarrow\rangle\top \vee \langle\rightarrow\rangle x),\\
& \text{in } p \wedge (\neg\langle\downarrow\rangle\top \vee \langle\downarrow\rangle x).
\end{aligned}$$

Theorem 8 (Reasoning on regular trees [4, 5])**.** *Given any two regular tree expressions e_1 and e_2, we have that for any tree $\mathcal{T}$ and valuations V and V':*

— $[\![e_1]\!]_V = \emptyset$*, if and only if,* $[\![F(e)]\!]^{\mathcal{T}}_{V'} = \emptyset$*;*

— $[\![e_1]\!]_V \subseteq [\![e_2]\!]_V$*, if and only if,* $[\![F(e_1) \wedge \neg F(e_2)]\!]^{\mathcal{T}}_{V'} = \emptyset$*; and*

— $F(e_i)$ *has linear size with respect to* e_i *(*$i = 1, 2$*).*

4.2 Interleaving

The interleaving operator, sometimes called shuffle operator, is a common extension of regular languages [27]. In particular, there is an interleaving operator in XML Schema and RelaxNG. Intuitively, the interleave of two regular tree expressions matches the concatenation of trees corresponding to the expressions regardless their order. This operator does not introduce more expressive power to regular languages, however, it is double-exponentially more succinct [19]. For instance, the interleaving of expressions pq and rs can be described as follows:

$$pq\&rs := pqrs + prqs + prsq + rpqs + rpsq + rspq.$$

Definition 16 (Interleaving)**.** The interleaving operator in regular tree expressions is inductively defined as follows:

$$\begin{aligned} e\&\epsilon &:= e,\\ e\epsilon\&e &:= e,\\ e_0\&(e_1+e_2) &:= (e_0\&e_1)+(e_0\&e_2),\\ (e_1+e_2)\&e_0 &:= (e_0\&e_1)+(e_0\&e_2),\\ (p_1[e_1]\cdot e_2)\&(p_2[e_3]\cdot e_4) &:= p_1[e_1]\cdot(e_2\&(p_2[e_3]\cdot e_4)),\\ &\quad + p_2[e_3]\cdot((p_1[e_1]\cdot e_2)\&e_4). \end{aligned}$$

Counting formulas in μTLIC can be used to represent the interleaving of regular tree expressions. For this purpose, we restrict the expressions that can be interleaved. This restriction is defined by the following grammar:

$$e' := p[e] \mid e'\cdot e' \mid e'+e',$$

where e is a regular tree expression without restrictions (Definition 13), and disjunctions have constant size, that is, for expressions $e'_1+e'_2$, we have that $|e'_1|^* = |e'_2|^*$, where:

$$\begin{aligned} |p[e]|^* &= 1,\\ |e_1\cdot e_2|^* &= |e_1|^* + |e_2|^*,\\ |e_1+e_2|^* &= \max\left(|e_1|^*, |e_2|^*\right). \end{aligned}$$

For instance, expressions of the form $p\&q^\star$ are disallowed, notice however that recursion can occur at another level of interleaving, for instance, $p\&r[q^\star]$. Now for an example of constant size disjunctions, $(ppp+qq)\&rrr$ is not allowed, because $|ppp|^* \neq |qq|^*$. Instead, equally sized disjunction can occur at the same level of interleaving, for example $(ppp+qqq)\&rrr$. Notice this restriction applies at top level only, hence expressions as the following are perfectly allowed $s[ppp+qq]\&rrr$.

We then define the translation of interleaving as follows.

Definition 17 (Translation of interleaving)**.** Given two regular tree expressions e'_1 and e'_2, we translate the interleaving operator as follows:

$$\begin{aligned} F(e'_1\&e'_2) := &\left(F(e'_1)=1\right)\wedge\left(F(e'_2)=1\right),\\ &\wedge\left(\top = chsize(e'_1\&e'_2)\right), \end{aligned}$$

$F(e'_i)$ is a linear translation of expression e'_i into a μTLIC formula:

$$\begin{aligned} F(p[e]) &:= p\wedge F^{\downarrow}(e),\\ F(e_1\cdot e_2) &:= F(e_1)\wedge\mu x.\langle\rightarrow\rangle(F(e_2)\vee x),\\ F(e_1+e_2) &:= F(e_1)\vee F(e_2). \end{aligned}$$

The translation F of unrestricted regular tree expressions is given in Definition 15, and $chsize$ is defined as follows:

$$\begin{aligned} chsize(p[e]) &= 1,\\ chsize(e_1+e_2) &= \max\left(chsize(e_1), chsize(e_2)\right),\\ chsize(e_1\cdot e_2) &= chsize(e_1\&e_2),\\ &= chsize(e_1)+chsize(e_2). \end{aligned}$$

Intuitively, $chsize$ computes the number of children to be interleaved.

As an example consider the expression $p[qr\&st]$. This can be expressed in terms of μTLIC as follows:

$$\begin{aligned} F(p[qr\&st]) &:= p\wedge[F(qr)=1]\wedge[F(st)=1],\\ &\quad\wedge[\top = chsize(qr\&st)],\\ &=: p\wedge[(q\wedge\mu x.\langle\rightarrow\rangle(r\vee x))=1],\\ &\quad\wedge[(s\wedge\mu x.\langle\rightarrow\rangle(t\vee x))=1]\wedge[\top=4]. \end{aligned}$$

Note that concatenation order is preserved, that is, q goes always first than r, and s goes first than t. However, none other order restriction is imposed, hence, p, q, r, s may occur interleaved, as long as we know there are only 4 children.

From Theorem 8 and Definition 17, it is now easy to see we can efficiently reason on regular expressions with interleaving in terms of the satisfiability of μTLIC formulas.

Theorem 9 (Reasoning on regular trees with interleaving)**.** *Given any two regular tree expressions e_1 and e_2 with interleaving, we have that for any tree $\mathcal{T}$ and valuations V and V' the following holds:*

- $[\![e_1]\!]_V = \emptyset$*, if and only if,* $[\![F(e_1)]\!]^{\mathcal{T}}_{V'} = \emptyset$*,*
- $[\![e_1]\!]_V \subseteq [\![e_2]\!]_V$*, if and only if,* $[\![F(e_1)\wedge\neg F(e_2)]\!]^{\mathcal{T}}_{V'} = \emptyset$*; and*

— $F(e_i)$ *has linear size with respect to* e_i *(*$i = 1, 2$*).*

Proof. For the first item, the proof goes by induction on the structure of e_1. All cases are identical as in Theorem 8. We only show here the case of interleaving, that is, when e_1 has the form $e'_1 \& e'_2$. Recall that:

$$F(e'_1 \& e'_2) := (F(e'_1) = 1) \wedge (F(e'_2) = 1)\,, \\ \wedge\, (\top = chsize(e'_1 \& e'_2))\,.$$

Now, it is proved by induction that $F(e'_i)$ is the translation of e'_i ($i = 1, 2$), that is, $[\![e'_i]\!]_V = \emptyset$, if and only if, $[\![F(e'_i)]\!]^{\mathcal{T}}_{V'} = \emptyset$. Consider e'_i is of the form $p[e]$, then:

$$F(p[e]) := p \wedge F^{\downarrow}(e).$$

The argument in this case also goes as the correspoding case of Theorem 8. Consider now this case:

$$F(e'_{i,1} \cdot e'_{i,2}) := F(e'_{i,1}) \wedge \mu x.\langle\rightarrow\rangle(F(e'_{i,2}) \vee x).$$

Which is is immediate since $F(e'_{i,j})$ (for $j = 1, 2$) corresponds by induction to the translation of $e'_{i,j}$. The case for $e'_{i,1} + e'_{i,2}$ also goes straightforward by induction.

Now, $F(e'_i) = 1$ then states $F(e'_i)$ occurs as a child only once. Nevertheless, there is no children order restriction. Since the number of children to be interleaved is constant, then $\top = chsize(e'_1 \& e'_2)$ fix the number of children to be interleaved. Therefore:

$$[\![e'_1 \& e'_2]\!]_V \neq \emptyset, \text{ if and only if, } [\![F(e'_1 \& e'_2)]\!]_V \neq \emptyset.$$

The second item is analogous. And the third one is straightforward by noticing F does not introduce duplications. □

4.3 Counting

Counting operators in regular languages restrict the occurrences of expressions with respect to natural numbers. For instance, $p^{[2,5]}$ denotes the finite concatenation of at least 2 p's and at most 5, this can be expressed as follows:

$$pp + ppp + pppp + ppppp.$$

As one may easily notice, this counting operators do not provided more expressive power, however, they are exponentially more succinct [19], that is, expressing $p^{[a,b]}$, where a and b are natural numbers encoded in binary, results in an exponentially larger regular expression (without counting constructors).

One may think that counting restrictions in regular languages may be easily expressed by counting formulas in μTLIC, however, recall that counting formulas do not impose any occurrence order, whereas counting restrictions in regular languages do, expressions must be consecutively concatenated. In contrast with counting regular expressions in [4, 5], where there is no order preservation, here we show that counting Presburger formulas may impose order restrictions on counting regular expressions, as in [28, 19]. Furthermore, in the current work, we consider a more general form of the counting than [28, 19], because counting expressions may no exhibit an upper bound.

Definition 18 (Counting regular tree expressions)**.** Counting regular tree expressions are defined as follows:

$$p\left[e'^{[a,b]}\right],$$

where e' is a regular expression without recursion at top level, that is, $e' := p[e] \mid e' \cdot e' \mid e' + e'$, where e is a regular expression without restrictions, a is a natural number encoded in binary, and b is also a natural number greater than a encoded in binary or ∞.

Intuitively, $e^{[a,b]}$ stands for the successive concatenation of e, such that it occurs at least a times and at most b times. If b is ∞, then there is no upper bound.

This can be see as a generalization of the Kleene star, which can be expressed by $e^{[0,\infty]}$. It is then worth to notice that although the general recursion operator is not allowed to occur inside the top level of the counting operator, other forms of recursion, as seen above with the Kleene star can be used. It is also important to note that with this subtle extension, allowing no upper bound, counting expressions become exponentially more

concise. This can be seen when expressing $e^{[a,\infty]}$, which can be encoded as $e^{[a,a]}e^{\star}$. The exponential gain becomes more evident when this duplication of e occurs in expresions with nested counting.

Definition 19 (Translation of counting)**.** We translate the counting operator as follows:

$$F\left(p\left[e'^{[a,b]}\right]\right) := p \wedge (a \leq \top \leq b) \wedge (\top = F(e')),$$

where e' is a regular expression without recursion at top level.

Consider as an example the following expression: $q\left[p^{[2,5]}\right]$. This can be expressed in terms of μTLIC formulas as follows:

$$q \wedge (2 \leq \top \leq b) \wedge (\top = p).$$

This formula means that q nodes have at least 2 p children, but no more than 5.

From Theorem 8 and Definition 19 we clearly can imply reasoning on regular expressions with counting and interleaving in terms of the satisfiability of μTLIC formulas. One may have noticed that the translation of counting regular tree expressions is not linear when considering the resulting counting formula as syntactic sugar. We then consider μTLIC extended in the obvious way with the additional counting operators. In Section 3, we present a satisfiability algorithm for μTLIC that can be easily extended with the syntactic sugar operator for counting.

Theorem 10 (Reasoning on regular trees with interleaving and counting)**.** *Given any two regular tree expressions e_1 and e_2 with interleaving and counting operators, we have that for any tree $\mathcal{T}$ and valuations V and V' the following holds:*

- $[\![e_1]\!]_V = \emptyset$*, if and only if,* $[\![F(e)]\!]^{\mathcal{T}}_{V'} = \emptyset$*;*
- $[\![e_1]\!]_V \subseteq [\![e_2]\!]_V$*, if and only if,* $[\![F(e_1) \wedge \neg F(e_2)]\!]^{\mathcal{T}}_{V'} = \emptyset$*; and*
- $F(e_i)$ *has linear size with respect to* e_i *(*$i = 1, 2$*).*

Proof. For the first item, the proof goes by induction on the structure of e_1. All cases are identical as in Theorem 8. The case of interleaving was shown in Theorem 9. Here, we only show the case of counting: $\left[\!\left[p\left[e'^{[a,b]}\right]\right]\!\right]_V = \emptyset$, if and only if, $[\![p \wedge (a \leq \top \leq b) \wedge (\top = F(e'))]\!]^{\mathcal{T}}_{V'} = \emptyset$. It is shown by induction that $F(e')$ corresponds to the translation of e':

$$[\![e']\!]_V = \emptyset, \text{ if and only if, } [\![F(e')]\!]^{\mathcal{T}}_{V'} = \emptyset.$$

This was already showed in the proof of Theorem 9. Then, it follows that $\top = F(e')$ restricts all children to match $F(e')$. $a \leq \top \leq b$ in addition constrain the number of children to be at least a but no more than b.

The second item is analogous. And the third one is straightforward by noticing F does not introduce duplications. □

5 Regular Counting Paths

XPath is a query language for semi-structured data (XML), its navigation core is known as regular paths, and it corresponds to the First Order Logic with two variables FO^2 [26]. We now introduce an extension of regular paths, considered in the specification of XPath [36], consisting of Presburger arithmetical constraints on children paths, that is, regular paths expressing children relations. We also give a new EXPTIME bound for reasoning on this counting extension of regular paths.

Definition 20 (Counting paths syntax)**.** We inductively define regular paths expressions with Presburger constraints by the following grammar:

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

where p is a proposition and a and b are integers encoded in binary, b is non-negative.

In order to ensure decidability, path expressions occuring in the scope of a counting operator $>$ are

restricted to children, that is, they are expression of the forms: $\downarrow, \downarrow\colon p, \downarrow [\beta]$, or $\downarrow\colon p[\beta]$.

We now give a formal description of the interpretation of regular paths with Presburger constraints.

Definition 21 (Counting paths semantics)**.** Given a tree structure $\mathcal{T}$, regular paths with Presburger constraints are interpreted as follows:

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

where $n_1 \xrightarrow{\alpha} n_2$ holds, if and only if, n_1 is related to n_2 through α in $\mathcal{T}$, and the interpretation of qualifiers (β) is the following.

$$\begin{aligned}
[\![\kappa > b]\!]^{\mathcal{T}} =& \left\{ n \mid \|\kappa\|^{\mathcal{T}_n} > b \right\},\\
\|a\zeta\|^{\mathcal{T}_n} =& a * \left| \left\{ n_1 \mid (n,n_1) \in [\![\zeta]\!]^{\mathcal{T}} \right\} \right|,\\
\|\kappa_1 + \kappa_2\|^{\mathcal{T}_n} =& \|\kappa_1\|^{\mathcal{T}_n} + \|\kappa_2\|^{\mathcal{T}_n},\\
[\![\varrho]\!]^{\mathcal{T}} =& \{n_1 \mid (n_1,n_2) \in [\![\varrho]\!]^{\mathcal{T}}\},\\
[\![\neg\beta]\!]^{\mathcal{T}} =& \mathcal{N} \setminus [\![\beta]\!]^{\mathcal{T}},\\
[\![\beta_1 \vee \beta_2]\!]^{\mathcal{T}} =& [\![\beta_1]\!]^{\mathcal{T}} \cup [\![\beta_2]\!]^{\mathcal{T}}.
\end{aligned}$$

Intuitively, regular paths are interpreted over tree structures as pairs of nodes. The left nodes, known as the context, represent from where the path is evaluated, and the right nodes, denote the selection of the path. Axis relation $\downarrow$, as in μTLIC, stands for the children relation, $\rightarrow$ for the following sibling relation, $\uparrow$ for parents, $\leftarrow$ for previous siblings, $\downarrow^\star$ for descendants, and $\uparrow^\star$ for ancestors.

So basic paths $\alpha : p$ denotes pair of nodes, such that the right node of the pair is labeled by p and it is related with the left node of the pair by α. So for instance, $\downarrow^\star\colon p$ stands for the pairs of nodes, such that the right node of the pairs is the descendant of the left node of the pair. ϱ/ϱ stands for the compositions of paths. For example, $\downarrow\colon p/\downarrow^\star\colon q$ intuitively navigates first to the children named p, and from there to the q descendants. $\varrho[\beta]$ denotes the pair of nodes of ϱ that satisfies β, which is a Boolean expression composed by regular paths and Presburger children paths. Consider for instance the path $\downarrow\colon p[\downarrow^\star\colon q]$, in contrast with the previous example, this expressions denotes the p children having at least one descendant named q. In Presburger expressions $\kappa > b$, path occurring in κ are children paths. For example, $\downarrow\colon p[\downarrow\colon q - \downarrow\colon r > 0]$ denotes the p children with more q children than r children. Another example is $\downarrow\colon p[\downarrow\colon q > 5]$, which denotes the p children with more than 5 q children. We also use the following syntactic sugar for qualifiers:

$$\begin{aligned}
\beta_1 \wedge \beta_2 :=& \neg(\neg\beta_1 \vee \neg\beta_2),\\
(\kappa \leq b) :=& \neg(\kappa > b),\\
(\kappa = k) :=& (\kappa \geq k) \wedge (\kappa > b - 1),\\
a_1\varrho_1 \# a_2\varrho_2 :=& (a_1\varrho + (-a_2)\varrho_2 \# 0),
\end{aligned}$$

where $\#$ stands for $<, \leq, \geq, =$. $/\rho$ stands for the pair of nodes denoted by ρ, such that the left node of the pairs is the root. Union, intersection and difference of paths are expressed as $\rho \cup \rho'$, $\rho \cap \rho'$, $\rho \setminus \rho'$, respectively.

Regular paths can be linearly translated in terms of μ-calculus [4]. Consider for instance the following expression: $\downarrow\colon p[\downarrow^\star\colon q]$. This path, evaluated from any node (context), selects the p children with at least one q descendant. Nodes selected by this path can be expressed by the following formula:

$$(p \wedge \langle\uparrow\rangle\top) \wedge \mu x.\langle\downarrow\rangle(q \vee x).$$

Arithmetical constraints on children path can be expressed by μTLIC counting expressions. For example, $\downarrow\colon p[\downarrow\colon q > b]$ selects the p nodes with at least $b+1$ children named q. This can be easily written in terms of μTLIC formulas as follows:

$$(p \wedge \langle\uparrow\rangle\top) \wedge (q > b).$$

As another example consider $\downarrow\colon p[\downarrow\colon q = \downarrow\colon r]$, which selects the p children with the same number of children named q and r. In terms of μTLIC, we then write:

$$(p \wedge \langle\uparrow\rangle\top) \wedge (q = r),$$

When characterizing regular paths in terms of μTLIC formulas, we can denote the context from where paths are evaluated by some other formula. We usually denote this context formula by C.

Definition 22 (Translation of counting paths)**.** We define the translation of regular paths with Presburger constraints, with respect to a context formula C, as follows:

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

where the translation of qualifiers F' is defined as follows.

$$\begin{aligned}
F'(\kappa > b, C) &:= F'(\kappa) > b,\\
F'(a\varrho, C) &:= aF''(\varrho),\\
F'(\kappa_1 + \kappa_2, C) &:= F'(\kappa_1) + F'(\kappa_2),\\
F'(\alpha, C) &:= F(\overline{\alpha}, C),\\
F'(\alpha : p, C) &:= F(\overline{\alpha} : p, C),\\
F'(\varrho_1/\varrho_2, C) &:= F'(\varrho_1, F'(\varrho_2, C)),\\
F'(\varrho[\beta], C) &:= F'(\varrho, F'(\beta, \top) \wedge C),\\
F'(\neg\beta, C) &:= \neg F'(\beta, C),\\
F'(\beta_1 \vee \beta_2, C) &:= F'(\beta_1, C) \vee F'(\beta_2, C),
\end{aligned}$$

where $\overline{\alpha}$ is the dual of α, that is, $\overline{\downarrow} = \uparrow$, $\overline{\Rightarrow} = \leftarrow$, $\overline{\downarrow^\star} = \uparrow^\star$, and $\overline{\overline{\alpha}} = \alpha$, and F'' translates children paths as follows:

$$\begin{aligned}
F''(\downarrow) &:= \top, & F''(\downarrow\colon p) &:= p,\\
F''(\downarrow[\beta]) &:= F'(\beta, \top), & F''(\downarrow\colon p[\beta]) &:= p \wedge F'(\beta, \top).
\end{aligned}$$

From this translation, it is then clear that μTLIC can be used as a query reasoning framework for regular paths with Presburger constraints.

Theorem 11 (Counting paths reasoning)**.** *For any regular path query with Presburger constraints ρ_1 and ρ_2, any formula C, any tree structure $\mathcal{T}$, and any valuation V, the following holds:*

- $[\![F(\rho_1, C)]\!]^{\mathcal{T}}_V = \left\{n \mid (n', n) \in [\![\rho_1]\!]^{\mathcal{T}}, n' \in [\![C]\!]^{\mathcal{T}}_V\right\}$;
- $[\![\rho_1]\!]^{\mathcal{T}} \subseteq [\![\rho_2]\!]^{\mathcal{T}}$ *if and only if* $[\![F(\rho_1, \top) \wedge \neg F(\rho_2, \top)]\!]^{\mathcal{T}}_V = \emptyset$*; and*
- $F(\rho_i)$ *has linear size with respect to* ρ_i *($i = 1, 2$).*

Proof. The proof of the first item goes by induction on the structure of the input query. Base cases are immediate, as well as most inductive ones. We will only consider then the case when the input query has the following form: $\varrho[\kappa > b]$. We then use another induction on the structure of κ. Consider then the case $\varrho[a \downarrow\colon p > b]$. According to Definition 22, we obtain the following:

$$F(\varrho[\downarrow\colon p > b, C]) := F(\varrho, C) \wedge (ap > b).$$

Now, by induction, we know that for any tree $\mathcal{T}$ and valuation V, it is the case that:

$$[\![F(\varrho, C)]\!]^{\mathcal{T}}_V = \left\{n \mid (n', n) \in [\![\varrho]\!]^{\mathcal{T}}, n' \in [\![C]\!]^{\mathcal{T}}_V\right\}.$$

Since $ap > b$ holds in nodes with more than b children, it is then easy to see that:

$$[\![F(\varrho[\downarrow\colon p > b], C)]\!]^{\mathcal{T}}_V = \left\{n \mid (n', n) \in [\![\varrho[a \downarrow\colon p > b]]\!]^{\mathcal{T}} \; n' \in [\![C]\!]^{\mathcal{T}}_V\right\}.$$

Other base cases for κ are analogous. Consider now the following input query $\varrho[\kappa_1 - \kappa_2 > b]$. This is translated as follows:

$$F(\varrho, C) \wedge F'(\kappa_1) + F'(\kappa_2) > b.$$

As in the base cases, by structural induction on paths, we know that ϱ exactly corresponds to its translation. By structural induction on children paths, we then obtain that κ_1 and κ_2 also correspond to their respective translation. It is then clear to infer the following:

$$[\![F(\varrho[\kappa_1+\kappa_2>b], C)]\!]_V^{\mathcal{T}} = \Big\{ n \mid n' \in [\![C]\!]_V^{\mathcal{T}} \; (n', n) \in [\![\varrho[\kappa_1+\kappa_2>b]]\!]^{\mathcal{T}} \Big\}.$$

The second item is an immediate consequence of the first one.

Regarding the third item, since the translation does not introduce duplications (Definition 22), the proof goes straightforward by structural induction. □

Corollary 1 (Query reasoning in the presence of schemas)**.** *Given any two regular tree expressions e_1 and e_2 with interleaving and counting operators, any regular paths with Presburger constraints ϱ_1 and ϱ_2, and any formula C, we have that for any tree $\mathcal{T}$ and valuation V the following holds:*

- *A query ρ_1 is empty in the presence of a regular tree (schema) e_1, if and only if, $[\![F(\varrho_1, C) \wedge F(e_1)]\!]_V^{\mathcal{T}} = \emptyset$;*
- *a query ρ_1 in the presence of a regular tree (schema) e_1 is contained in a query ρ_2 in the presence of a regular tree e_2, if and only if,*

  $$[\![(F(\varrho_1, C) \wedge F(e_1)) \wedge \neg\, (F(\varrho_2, C) \wedge F(e_2))]\!]_V^{\mathcal{T}} = \emptyset;$$

  and
- $F(e_i)$ *and* $F(\varrho_i, C)$ *have linear size with respect to* e_i, ϱ_i *($i = 1, 2$), and* C.

6 Conclusions

We introduced a modal logic for trees with a fixed point, inverse programs, and Presburger constraints (μTLIC). This logic can been seen as the fully enriched μ-calculus for trees extended with Presburger constraints.

Regular tree languages (XML schemas) can be linearly captured by the logic. We introduced extensions of regular trees with interleaving and counting operators. These extensions can also be linearly characterized by μTLIC. Moreover, regular path queries (XPath) with Presburger constraints on children paths are also linearly translated in terms of μTLIC formulas.

Since the logic is closed under negation, it can be used as a XML reasoning framework for counting extensions of XPath and XML schemas. We showed that the logic is decidable in single exponential time, even if the Presbuger constraints are encoded in binary.

This result implies new EXPTIME bounds on XPath counting fragments and regular tree extensions with interleaving and counting.

In [6, 3], decidable classes of ranked trees with counting and (dis)equality constraints are studied. As a further research perspective, we are interested in the relation of counting and equality constraints on unranked trees. We believe efficient decidability algorithms may be extracted from the modal logic approach.

In another setting, arithmetical constraints on trees have been also successfully used in the verification of balanced tree structures such as AVL or red-black trees [25, 21].

We believe another field of application for the logic presented in the current work is in the verification of balanced tree structures. We also believe the logic can be used as an expressive framework in context-aware systems [7, 23], where counting constraints play a key role when modeling location/distance variables.

References

1. **Aminof, B., Murano, A., & Rubin, S.** (**2018**). CTL* with graded path modalities. *Inf. Comput.*, 262(Part), 1–21.
2. **Areces, C., Hoffmann, G., & Denis, A.** (**2010**). Modal logics with counting. In **Dawar, A. & de Queiroz, R. J. G. B.**, editors, *Logic, Language, Information and Computation, WoLLIC 2010*, volume 6188 of *Lecture Notes in Computer Science*. Springer, 98–109.
3. **Bárcenas, E., Benítez-Guerrero, E., & Lavalle, J.** (**2016**). On regular paths with counting and data tests. *Electr. Notes Theor. Comput. Sci.*, 328, 3–16.
4. **Bárcenas, E., Genevès, P., Layaïda, N., & Schmitt, A.** (**2011**). Query reasoning on trees with types, interleaving, and counting. In **Walsh, T.**, editor, *IJCAI*. IJCAI/AAAI, 718–723.
5. **Bárcenas, E. & Lavalle, J.** (**2014**). Global numerical constraints on trees. *Logical Methods in Computer Science*, 10(2).
6. **Barguñó, L., Creus, C., Godoy, G., Jacquemard, F., & Vacher, C.** (**2013**). Decidable classes of tree automata mixing local and global constraints modulo flat theories. *Logical Methods in Computer Science*, 9(2).
7. **Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., & Riboni, D.** (**2010**). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2), 161–180.
8. **Bianco, A., Mogavero, F., & Murano, A.** (**2012**). Graded computation tree logic. *ACM Trans. Comput. Log.*, 13(3), 25.
9. **Bonatti, P. A., Lutz, C., Murano, A., & Vardi, M. Y.** (**2006**). The complexity of enriched mu-calculi. In **Bugliesi, M., Preneel, B., Sassone, V., & Wegener, I.**, editors, *ICALP*, volume 4052 of *Lecture Notes in Computer Science*. Springer, 540–551.
10. **Bonatti, P. A., Lutz, C., Murano, A., & Vardi, M. Y.** (**2008**). The complexity of enriched mu-calculi. *Logical Methods in Computer Science*, 4(3).
11. **Bonatti, P. A. & Peron, A.** (**2004**). On the undecidability of logics with converse, nominals, recursion and counting. *Artif. Intell.*, 158(1), 75–96.
12. **Calvanese, D., Giacomo, G. D., Lenzerini, M., & Vardi, M. Y.** (**2010**). Node selection query languages for trees. In **Fox, M. & Poole, D.**, editors, *AAAI*. AAAI Press.
13. **Charatonik, W. & Witkowski, P.** (**2013**). Two-variable logic with counting and trees. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*. IEEE Computer Society, 73–82.
14. **Charatonik, W. & Witkowski, P.** (**2016**). Two-variable logic with counting and trees. *ACM Trans. Comput. Log.*, 17(4), 31:1–31:27.
15. **Colazzo, D., Ghelli, G., Pardini, L., & Sartiani, C.** (**2013**). Efficient asymmetric inclusion of regular expressions with interleaving and counting for XML type-checking. *Theor. Comput. Sci.*, 492, 88–116.
16. **Demri, S. & Lugiez, D.** (**2010**). Complexity of modal logics with Presburger constraints. *J. Applied Logic*, 8(3), 233–252.
17. **Droste, M. & Vogler, H.** (**2011**). Weighted logics for unranked tree automata. *Theory of Computing Systems*, 48(1), 23–47.
18. **Fischer, M. J. & Ladner, R. E.** (**1977**). Propositional modal logic of programs (extended abstract). In **Hopcroft, J. E., Friedman, E. P., & Harrison, M. A.**, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*. ACM, 286–294.
19. **Gelade, W.** (**2010**). Succinctness of regular expressions with interleaving, intersection and counting. *Theor. Comput. Sci.*, 411(31-33), 2987–2998.
20. **Genevès, P., Layaïda, N., Schmitt, A., & Gesbert, N.** (**2015**). Efficiently deciding μ-calculus with converse over finite trees. *ACM Trans. Comput. Log.*, 16(2), 16.
21. **Habermehl, P., Iosif, R., & Vojnar, T.** (**2010**). Automata-based verification of programs with tree updates. *Acta Inf.*, 47(1), 1–31.
22. **Hosoya, H., Vouillon, J., & Pierce, B. C.** (**2005**). Regular expression types for XML. *ACM Trans. Program. Lang. Syst.*, 27(1), 46–90.
23. **Limón, Y., Bárcenas, E., Benítez-Guerrero, E., & Molero, G.** (**2018**). On the consistency of context-aware systems. *Journal of Intelligent and Fuzzy Systems*, 34(5), 3373–3383.
24. **Malvone, V., Mogavero, F., Murano, A., & Sorrentino, L.** (**2018**). Reasoning about graded strategy quantifiers. *Inf. Comput.*, 259(3), 390–411.
25. **Manna, Z., Sipma, H. B., & Zhang, T.** (**2007**). Verifying balanced trees. In **Artëmov, S. N. & Nerode, A.**, editors, *LFCS*, volume 4514 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-540-72732-3, 363–378.

26. Marx, M. (**2005**). Conditional XPath. *ACM Trans. Database Syst.*, 30(4), 929–959.

27. Mayer, A. J. & Stockmeyer, L. J. (**1994**). Word problems-this time with interleaving. *Inf. Comput.*, 115(2), 293–311.

28. Meyer, A. R. & Stockmeyer, L. J. (**1972**). The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*. IEEE Computer Society, 125–129.

29. Murata, M., Lee, D., Mani, M., & Kawaguchi, K. (**2005**). Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4), 660–704.

30. Papadimitriou, C. H. (**1981**). On the complexity of integer programming. *J. ACM*, 28(4), 765–768.

31. Seidl, H., Schwentick, T., & Muscholl, A. (**2003**). Numerical document queries. In **Neven, F., Beeri, C., & Milo, T.**, editors, *PODS*. ACM. ISBN 1-58113-670-6, 155–166.

32. Seidl, H., Schwentick, T., & Muscholl, A. (**2008**). Counting in trees. In **Flum, J., Grädel, E., & Wilke, T.**, editors, *Logic and Automata*, volume 2 of *Texts in Logic and Games*. Amsterdam University Press, 575–612.

33. Seidl, H., Schwentick, T., Muscholl, A., & Habermehl, P. (**2004**). Counting in trees for free. In **Díaz, J., Karhumäki, J., Lepistö, A., & Sannella, D.**, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*. Springer, 1136–1149.

34. Sorrentino, L., Rubin, S., & Murano, A. (**2018**). Graded CTL* over finite paths. In **Aldini, A. & Bernardo, M.**, editors, *Proceedings of the 19th Italian Conference on Theoretical Computer Science*, volume 2243 of *CEUR Workshop Proceedings*. CEUR-WS.org, 152–161.

35. Tarski, A. (**1955**). A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5(2), 285–309.

36. ten Cate, B. & Marx, M. (**2009**). Axiomatizing the logical core of XPath 2.0. *Theory Comput. Syst.*, 44(4), 561–589.

37. Tobies, S. (**2001**). *Complexity results and practical algorithms for logics in knowledge representation*. Ph.D. thesis, RWTH Aachen University, Germany.

38. Venema, Y. (**2012**). *Lecture Notes on the modal μ-calculus*. The University of Amsterdam.

39. Zawidzki, M., Schmidt, R. A., & Tishkovsky, D. (**2013**). Satisfiability problem for modal logic with global counting operators coded in binary is NExpTime-complete. *Inf. Process. Lett.*, 113(1-2), 34–38.

Article received on 27/04/2018; accepted on 29/10/2019.
Corresponding author is Everardo Bárcenas.