A Tree Logic with Graded Paths and Nominals

Everardo Barcenas, Pierre Genevès, Nabil Layaïda, Alan Schmitt

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Tree Logic with Graded Paths and Nominals

Everardo Bárcenas — Pierre Genevès — Nabil Layaïda — Alan Schmitt

N° 7251

April 2010

Knowledge and Data Representation and Management

ISSN 0249-6399 ISRN INRIA/RR--7251--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

A Tree Logic with Graded Paths and Nominals

Everardo Bárcenas , Pierre Genevès , Nabil Layaïda , Alan Schmitt

Theme : Knowledge and Data Representation and Management
Équipes-Projets WAM et SARDES

Rapport de recherche n° 7251 — April 2010 — 34 pages

Abstract: Regular tree grammars and regular path expressions constitute core constructs widely used in programming languages and type systems. Nevertheless, there has been little research so far on reasoning frameworks for path expressions where node cardinality constraints occur along a path in a tree. We present a logic capable of expressing deep counting along paths which may include arbitrary recursive forward and backward navigation. The counting extensions can be seen as a generalization of graded modalities that count immediate successor nodes. While the combination of graded modalities, nominals, and inverse modalities yields undecidable logics over graphs, we show that these features can be combined in a tree logic decidable in exponential time.

Key-words: Modal Logic, XML, XPath, Schema

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

A Tree Logic with Graded Paths and Nominals

Résumé : Ce document introduit une logique d'arbre décidable en temps exponentielle et qui est capable d'exprimer des contraintes de cardinalité sur chemins multidirectionnelle

Mots-clés : Logique Modal, XML, XPath, Schema

1 Introduction

A fundamental peculiarity of XML is the description of regular properties. For example, in XML schema languages the content types of element definitions is made through the use of regular expressions. In addition, selecting nodes in such constrained trees is also done by the mean of regular path expressions (à la XPath). In both cases, it is often interesting to be able to express conditions on the frequency of occurrences of nodes.

Even if we consider simple strings, it is well known that some formal languages easily described in English may require voluminous regular expressions. For instance, as pointed in [HJJ+95], the language L_{2a2b} of all strings over $\Sigma = \{a, b, c\}$ containing at least two occurrences of a *and* at least two occurrences of b seems to require a large expression, such as:

$$\begin{array}{ccccc} & \Sigma^*a\Sigma^*a\Sigma^*b\Sigma^*b\Sigma^* & \cup & \Sigma^*a\Sigma^*b\Sigma^*a\Sigma^*b\Sigma^* \\ \cup & \Sigma^*a\Sigma^*b\Sigma^*b\Sigma^*a\Sigma^* & \cup & \Sigma^*b\Sigma^*b\Sigma^*a\Sigma^*a\Sigma^* \\ \cup & \Sigma^*b\Sigma^*a\Sigma^*b\Sigma^*a\Sigma^* & \cup & \Sigma^*b\Sigma^*a\Sigma^*a\Sigma^*b\Sigma^*. \end{array}$$

If we added $\cap$ to the operators for forming regular expressions, then the language $\{a, b, c\}$ could be expressed more concisely as $(\Sigma^*a\Sigma^*a\Sigma^*) \cap (\Sigma^*b\Sigma^*b\Sigma^*)$. In logical terms, conjunction offers a first dramatic reduction in expression size.

If we now consider a formalism equipped with the ability of describing numerical constraints on the frequency of occurrences, we get a second (exponential) reduction in size. For instance, the above expression can be formulated as $(\Sigma^*a\Sigma^*)^2 \cap (\Sigma^*b\Sigma^*)^2$. We can even write $(\Sigma^*a\Sigma^*)^{2^{20}} \cap (\Sigma^*b\Sigma^*)^{2^{20}}$ instead of a (much) larger expression.

Different extensions of regular expressions with intersection, counting constraints, and interleaving have been recently considered over strings, and for describing content models of sibling nodes in XML type languages [CGS09, GMN08, KT07]. The complexity of the inclusion problem over these different language extensions and their combinations typically ranges from polynomial to exponential space (see [GMN08] for a survey). The main distinction between these works and the work presented here is that we focus on counting nodes located along deep and recursive paths in trees.

When considering regular *tree* languages instead of regular *string* languages, succinct syntactic sugars such as the ones presented above are even more useful, as branching makes the situation more combinatorial compared to strings. In the case of trees, it is often useful to express cardinality constraints not only on the sequence of children nodes, but also in a particular region of a tree: in a subtree for example. Suppose for instance that we want to define a tree language over Σ where there is no more than 2 "b" nodes. This seems to require a quite

large regular tree type expression such as the one below:

$$\begin{array}{lcl} x_{\text{root}} & \rightarrow & b[x_{b\leq 1}] \mid c[x_{b\leq 2}] \mid a[x_{b\leq 2}] \\ x_{b\leq 2} & \rightarrow & x_{\neg b}, b[x_{\neg b}], x_{\neg b}, b[x_{\neg b}], x_{\neg b} \mid x_{\neg b}, b[x_{b\leq 1}], x_{\neg b} \\ & & \mid x_{\neg b}, a[x_{b\leq 2}], x_{\neg b} \mid x_{\neg b}, c[x_{b\leq 2}], x_{\neg b} \mid x_{b\leq 1} \\ x_{b\leq 1} & \rightarrow & x_{\neg b} \mid x_{\neg b}, b[x_{\neg b}], x_{\neg b} \mid a[x_{b\leq 1}] \mid c[x_{b\leq 1}] \\ x_{\neg b} & \rightarrow & (a[x_{\neg b}] \mid c[x_{\neg b}])^* \end{array}$$

where x_{root} is the starting non-terminal; $x_{\neg b}, x_{b\leq 1}, x_{b\leq 2}$ are non-terminals; and the bracket notation $a[x_{\neg b}]$ describes a subtree whose root is labeled a and in which there is no b node.

More generally, the widely adopted notations for regular tree grammars produce very verbose definitions for properties involving cardinality constraints on the nesting of elements[1].

The problem with regular tree (and even string) grammars is that one is forced to fully expand all the patterns of interest using concatenation, union, and Kleene star. Instead, it is often tempting to rely on another kind of (formal) notation that just describes a simple pattern and additional constraints on it. For instance, one could imagine denoting the previous example as follows, where the additional constraint is described using XPath notation:

$$(x \rightarrow (a[x] \mid b[x] \mid c[x])^*) \ \wedge \ \text{count}(/\text{descendant-or-self::}b) \leq 2$$

Although this kind of counting operators does not increase the expressive power of the regular tree grammars, they can have a drastic impact on succinctness, thus making reasoning over these languages harder (as noticed in [Gel08] in the case of strings). Indeed, reasoning on this kind of extensions without relying on their expansion (in order to avoid syntactic blow-ups) is often tricky [GGM09]. Determining satisfiability, containment, and equivalence over these classes of extended regular expressions typically require involved algorithms with extra-complexity [MS72] compared to plain vanilla regular expressions.

In the present paper, we propose a logical notation that happens to be especially appropriate for describing many sorts of cardinality constraints on the frequency of occurrence of nodes in regular tree types. Regular tree types encompass most of XML types (DTDs, XML Schemas, RelaxNGs) used in practice today.

XPath is the standard query language for XML documents, and it is an important part of other XML technologies such as XSLT and XQuery. XPath expressions are regular path expressions interpreted as sets of nodes selected from a given context node. In contrast with regular tree types, which only express properties on children nodes, most of the expressive power of XPath comes from the ability to perform multidirectional navigation, that is, XPath expressions are able to express properties involving not only recursive navigation, as

[1]This is typically the reason why the standard DTD for XHTML does not syntactically prevent the nesting of anchors, whereas this nesting is actually prohibited in the XHTML standard.

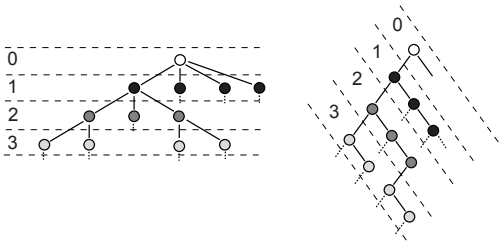

Figure 1: n–ary to binary trees

for descendant nodes for instance, but also backward navigation, as for ancestor nodes. Unfortunately, expressing cardinality restrictions on nodes accessible by recursive multidirectional paths may introduce an extra-exponential cost [GR05, tCM09], or may even lead to undecidable formalisms [tCM09, DL06]. We propose in this paper a decidable framework capable of succinctly express cardinality constraints along deep multidirectional paths.

Contribution and Outline We introduce a tree logic with counting operators for expressing arbitrarily deep and recursive counting constraints in Section 2. A sound and complete algorithm for testing satisfiability of logical formulas in exponential time is presented in Section 3. Section 4 shows how the logic and the algorithm can be applied in the XML setting and in particular for the static analysis of XPath expressions and common schemas containing constraints on the frequency of occurrence of nodes. Finally, we review related works in Section 5 before concluding in Section 6.

2 Counting Tree Logic

We first present trees that we consider, and define a notion of trails in trees, before introducing the syntax and semantics of logical formulas.

2.1 Trees

We consider finite trees which are node-labeled and sibling-ordered. Since there is a well-known bijective encoding between n–ary and binary trees, we focus on binary trees without loss of generality. Specifically, we use the encoding represented in Figure 1, where the binary representation preserves the first child of a node and append sibling nodes as second successors.

We consider the modalities "$\triangledown$" and "$\triangleright$". The modality "$\triangledown$" labels the edge between a node and its first child. The modality "$\triangleright$" labels the edge between a node and its next sibling. We also consider the converse modalities "$\triangle$" and "$\triangleleft$" that respectively labels the same edges in the reverse direction.

In order to define a simple set theoretic semantics for the logic, we consider trees in a way similar to Kripke structures for modal logics [Var98]. Specifically, we name $M = \{\triangledown, \triangleright, \triangle, \triangleleft\}$ the set of *modalities*. For $m \in M$ we denote by $\overline{m}$ the corresponding inverse modality ($\overline{\triangledown} = \triangle, \overline{\triangleright} = \triangleleft, \overline{\triangle} = \triangledown, \overline{\triangleleft} = \triangleright$). We consider a countable alphabet P of *propositions* representing names of nodes. A node is labeled with exactly one proposition.

A tree can then be seen as a tuple (N, R, L), where: N is a finite set of nodes; R is a partial mapping from $N \times M$ to N that restricts the labeling of edges to form a tree structure; and L is a labeling function from N to P.

2.2 Trails

Trails are defined as regular expressions formed by modalities, as follows:

$$\alpha_0 ::= m \mid \alpha_0, \alpha_0 \mid \alpha_0 \shortmid \alpha_0$$
$$\alpha ::= \alpha_0 \mid \alpha_0^\star \alpha_0$$

We restrict trails to sequences or repeated subtrails (which contain no repetition) followed by a subtrail (with no repetition). We also disallow trails of the form $m, \overline{m}$, which may result in formulas with cycles.

The syntactic interpretation of trails corresponds to sets of sequences of modalities (as in the usual semantics of regular expressions).

In a given tree, we say that there is a *trail α from the node n_0 to the node n_k*, written $n_0 \xrightarrow{\alpha} n_k$, if and only if there is a sequence of nodes $n_0, \ldots, n_k$ and a sequence of modalities $m_1, \ldots, m_k$ that belongs to the syntactic interpretation of the trail α, such that $R(n_j, m_{j+1}) = n_{j+1}$, where $j = 0, \ldots, k-1$. We say that a path ρ among two nodes belongs to a trail α, written $\rho \in \alpha$, if there exists a sequence of modalities between the nodes that belongs to the interpretation of the trail.

2.3 Syntax of Logical Formulas

The syntax of logical formulas is given in Figure 2, where $m \in M$ and $k \in \mathbb{N}$. The syntax is shown in negation normal form, which can be reached usual De Morgan rules together with rules given in Figure 3. The fact that the semantic interpretation is preserved even though the smallest fixpoint does not become a greatest fixpoint is a consequence of Lemma 2.1.

Defining an *equality* operator for counting formulas is straightforward.

$$\langle\alpha\rangle_{=k}\psi \equiv \langle\alpha\rangle_{>(k-1)}\psi \wedge \langle\alpha\rangle_{\leq k}\psi \qquad \text{if } k > 0$$
$$\langle\alpha\rangle_{=0}\psi \equiv \langle\alpha\rangle_{\leq 0}\psi$$

2.4 Semantics of Logical Formulas

Formulas are interpreted as sets of nodes in a tree. A model of a formula is a tree, such that the formula denotes a non-empty set of nodes in this tree. A

$\Phi \ni \phi$	::=			formula
		$\top \mid \neg\top$		true, false
	$\mid$	$p \mid \neg p$		atomic prop (negated)
	$\mid$	x		recursion variable
	$\mid$	$\phi \vee \phi$		disjunction
	$\mid$	$\phi \wedge \phi$		conjunction
	$\mid$	$\langle m \rangle \phi$	$\mid \neg\langle m \rangle\top$	modality (negated)
	$\mid$	$\langle \alpha \rangle_{\leq k} \psi$	$\mid \langle \alpha \rangle_{>k} \psi$	counting
	$\mid$	$\mu x.\psi$		fixpoint operator
ψ	::=			
		$\top \mid \neg\top$		
	$\mid$	$p \mid \neg p$		
	$\mid$	x		
	$\mid$	$\psi \vee \psi$		
	$\mid$	$\psi \wedge \psi$		
	$\mid$	$\langle m \rangle \psi$	$\mid \neg\langle m \rangle\top$	
	$\mid$	$\mu x.\psi$		

Figure 2: Syntax of Formulas (in Normal Form).

$$\neg\langle m \rangle \phi \equiv \neg\langle m \rangle\top \vee \langle m \rangle\neg\phi \qquad \neg\mu x.\psi \equiv \mu x.\neg\psi\{^{x}/_{\neg x}\}$$
$$\neg\langle \alpha \rangle_{\leq k} \psi \equiv \langle \alpha \rangle_{>k} \psi \qquad \neg\langle \alpha \rangle_{>k} \psi \equiv \langle \alpha \rangle_{\leq k} \psi$$

Figure 3: Reduction to Negation Normal Form.

$$
\begin{array}{lcl}
[\![\top]\!]^T_V & = & N \\
[\![\neg\top]\!]^T_V & = & \varnothing \\
[\![p]\!]^T_V & = & \{n, L(n) = p\} \\
[\![\neg p]\!]^T_V & = & \{n, L(n) \neq p\} \\
[\![x]\!]^T_V & = & \{n, (n,x) \in V\} \\
[\![\phi_1 \vee \phi_2]\!]^T_V & = & [\![\phi_1]\!]^T_V \cup [\![\phi_2]\!]^T_V \\
[\![\phi_1 \wedge \phi_2]\!]^T_V & = & [\![\phi_1]\!]^T_V \cap [\![\phi_2]\!]^T_V \\
[\![\langle m\rangle\phi]\!]^T_V & = & \{n, R(n,m) \in [\![\phi]\!]^T_V\} \\
[\![\neg\langle m\rangle\top]\!]^T_V & = & \{n, R(n,m) \text{ undefined}\} \\
[\![\langle\alpha\rangle_{\leq k}\psi]\!]^T_V & = & \{n, |\{n' \in [\![\psi]\!]^T_V \mid n \xrightarrow{\alpha} n'\}| \leq k\} \\
[\![\langle\alpha\rangle_{>k}\psi]\!]^T_V & = & \{n, |\{n' \in [\![\psi]\!]^T_V \mid n \xrightarrow{\alpha} n'\}| > k\} \\
[\![\mu x.\psi]\!]^T_V & = & \bigcap\{N', [\![\psi]\!]^T_{V[^{N'}/_x]} \subseteq N'\}
\end{array}
$$

Figure 4: Semantics of Formulas.

counting formula $\langle\alpha\rangle_{>k}\psi$ is interpreted as follows: the set of nodes such that there are at least $k+1$ nodes satisfying ψ through the trail α. For example, the formula $p_1 \wedge \langle\triangledown\rangle\langle\triangleright^*\rangle_{>5}p_2$, denotes p_1 nodes with strictly more than 5 children nodes named p_2.

In order to present the formal semantics of formulas, we introduce valuations. Given a tree, a *valuation* V is a binary relation between tree nodes and variables. We write $V[^{N'}/_x]$, where N' is a subset of the nodes, for the relation denoted by V extended with (n,x) for every $n \in N'$. Given a tree $T = (N, R, L)$ and a valuation V, the formal semantics of formulas is given in Figure 4.

Intuitively, the formulas are interpreted as sets of nodes in a tree: propositions denote the nodes where they occur; negation is interpreted as set complement; disjunction and conjunction are respectively set union and intersection; the least fixpoint operator performs finite recursive navigation; and the counting operator denotes certain nodes, named the source nodes, such that the nodes, accessible from a single source through a trail, fulfill a cardinality restriction. A formula is said to be *satisfiable* when its interpretation is not empty.

2.5 Restriction over Formulas

We consider a syntactic restriction over formulas similar to the one in [GLS07]: every formula of the logic must be *cycle-free* (so that the logic is closed under negation [GLS07]). Intuitively, in a cycle-free formula, fixpoint variables do not occur in the scope of both a modality and its converse. For example, cycle-free trails are trails where both a subtrail and its converse do not occur under the

scope of the recursion operator. We do not consider counting formulas under fixpoints nor under counting formulas.

Lemma 2.1. *Let ϕ be a cycle-free formula, and T be a tree for which $[\![\phi]\!]^T_\varnothing \neq \varnothing$. Then there is a finite unfolding ϕ' of the fixpoints of ϕ such that $[\![\phi'\{{}^{\neg\top}/_{\mu x.\psi}\}]\!]^T_\varnothing = [\![\phi]\!]^T_\varnothing$.*

Proof. As counting formulas may be replaced by non-counting formulas (with the cost of an exponential blow up), the proof is identical to the one in [GLS07]. □

2.6 Global Counting Formulas and Nominals

An interesting consequence of the inclusion of backward axes in trails is the ability to reach every node in the tree from a given node of the tree, using the trail $(\triangle|\triangleleft)^\star, (\triangledown|\triangleright)^\star$[2]. We can thus select some nodes depending on some global counting property. Consider the following formula, where # stands for one of the comparison operators $\leq, >, =$.

$$\langle (\triangle|\triangleleft)^\star, (\triangledown|\triangleright)^\star \rangle_{\#k}\phi_1$$

Intuitively, this formula considers each node n of the tree, and counts how many nodes in the whole tree satisfy ϕ_1. It then selects node n if and only if the count is compatible with the comparison considered. This formula thus returns either every node of the tree, or the empty set. It is then easy to restrict the selected nodes to some that satisfy a given formula ϕ_2, using intersection.

$$\langle (\triangle|\triangleleft)^\star, (\triangledown|\triangleright)^\star \rangle_{\#k}\phi_1 \wedge \phi_2$$

This formula select every node satisfying ϕ_2 if and only if there are $\#k$ nodes satisfying ϕ_1, which we write as follows.

$$\phi_1 \# k \implies \phi_2$$

We can now express existential properties, such as "select all nodes satisfying ϕ_2 if there exists a node satisfying ϕ_1".

$$\phi_1 > 0 \implies \phi_2$$

We can also express universal properties, such as "select all nodes satisfying ϕ_2 if every node satisfies ϕ_1".

$$(\neg\phi_1) \leq 0 \implies \phi_2$$

Another way to interpret global counting formulas is as a generalization of the so-called nominals in the modal logics community [SV01]. Nominals are special propositions whose interpretation is a singleton (they occur exactly once

[2]Note that this trail is cycle-free.

in the model). They come for free with the logic. A nominal, denoted "@n" in the remaining part of the paper, corresponds to the following global counting formula:

$$\langle (\triangle|\lhd)^\star, (\triangledown|\rhd)^\star \rangle_{=1} n$$

where n is a new fresh atomic proposition.

Notice that we can also perform a navigation to everywhere in a tree with only fixpoint formulas, hence a nominal can be alternatively written as:

$$\begin{aligned} @n \equiv n \wedge \neg[&\text{descendant}(n) \vee \text{ancestor}(n) \vee \\ &\text{desc-or-self}(\text{siblings}(n)) \vee \\ &\text{desc-or-self}(\text{siblings}(\text{ancestor}(n)))], \end{aligned}$$

where:

$$\begin{array}{lcl} \text{descendant}(\phi) & = & \langle\triangledown\rangle \mu x.\phi \vee \langle\triangledown\rangle x \vee \langle\rhd\rangle x \\ \text{foll-sibling}(\phi) & = & \mu x.\langle\rhd\rangle \phi \vee \langle\rhd\rangle x \\ \text{prec-sibling}(\phi) & = & \mu x.\langle\lhd\rangle \phi \vee \langle\lhd\rangle x \\ \text{desc-or-self}(\phi) & = & \mu x_0.\phi \vee \langle\triangledown\rangle \mu x_1.x_0 \vee \langle\rhd\rangle x_1 \\ \text{ancestor}(\phi) & = & \mu x.\langle\triangle\rangle(\phi \vee x) \vee \langle\lhd\rangle x \\ \text{siblings}(\phi) & = & \text{foll-sibling}(\phi) \vee \text{prec-sibling}(\phi) \end{array}$$

2.7 Graded Paths

Graded modalities have been introduced to count immediate successor nodes in graphs [KSV02]. Specifically, graded modalities make it possible to restrict the number of occurrences of immediate successors of a node in a graph by the mean of an explicit constant upper-bound and/or lower-bound. Here we consider trees and extend the "immediate successor" notion to nodes reachable from any regular path, including reverse and recursive navigation.

A peculiarity of graded modalities in graphs is that they can be used inside recursive formulas. A similar notion in trees consists in counting immediate children nodes, as performed by the counting formula $\langle\triangledown\rangle\langle\rhd^*\rangle_{\#k}\phi$, where ϕ describes the property to be counted. It is then possible to consider occurrences of this counting formula inside a fixpoint operator. This is because this peculiar counting formula can be simply rewritten in terms of plain vanilla logical formulas. For instance, the formula $\langle\triangledown\rangle\langle\rhd^*\rangle_{>1}p$ states the existence of at least two "p" children, and is translated into:

$$\langle\triangledown\rangle \mu x.(p \wedge \langle\triangledown\rangle \mu y.p \vee \langle\rhd\rangle y) \vee \langle\rhd\rangle x$$

The general nesting scheme of this translation can be expressed as follows, where the function ch$(\cdot)$ takes such a counting formula as input and returns

its translation:

$$\begin{aligned}
\mathrm{ch}(\langle\triangledown\rangle\langle\triangleright^*\rangle_{>0}\phi) &= \langle\triangledown\rangle\mu x.\phi \vee \langle\triangleright\rangle x \\
\mathrm{ch}(\langle\triangledown\rangle\langle\triangleright^*\rangle_{>k+1}\phi) &= \langle\triangledown\rangle\mu x.\,(\phi \wedge \mathrm{ch}(\langle\triangledown\rangle\langle\triangleright^*\rangle_{>k}\phi)) \vee \langle\triangleright\rangle x \\
\mathrm{ch}(\langle\triangledown\rangle\langle\triangleright^*\rangle_{\leq k}\phi) &= \neg\mathrm{ch}(\langle\triangledown\rangle\langle\triangleright^*\rangle_{>k}\phi)
\end{aligned}$$

We can even apply a recursive version of this transformation in order to rewrite nested counting formulas.

In Lemma 3.12, we show the computational cost of the translation does not depend on the size of the formula, but on the nesting level of counting subformulas.

The possibility of using an arbitrary fixpoint operator around a given formula allows one to express the "until" operator, proposed for XPath by Marx [Mar05]. Owing to the previous translation, we can combine counting features with the "until" operator and express properties that go beyond the expressive power of the XPath 1.0 standard. For instance, the following formula states that "starting from the current node, until we reach an ancestor named a, every ancestor has at least 3 children named b":

$$\mu x.\,(\langle\triangledown\rangle\langle\triangleright^*\rangle_{>2}b \wedge \mu y.\langle\triangle\rangle x \vee \langle\triangleleft\rangle y) \vee a$$

3 Satisfiability Algorithm

We present a tableau-based algorithm for checking satisfiability of formulas. Given a formula, the algorithm seeks to build a satisfying tree. A satisfying tree is found if and only if the formula is satisfiable, otherwise the algorithm concludes that the formula is unsatisfiable.

3.1 Overview

The algorithm operates in two stages.

First, a formula ϕ is decomposed into a set of subformulas, called the *Lean*. The Lean gathers all subformulas that are useful for determining the truth status of the initial formula, while eliminating redundancies. For instance, conjunctions and disjunctions are eliminated at this stage, since, if a subformula ϕ_1 holds then one does not need to know the truth status of ϕ_2 in order to determine the truth status of $\phi_1 \vee \phi_2$. In fact, the lean (defined in 3.2) only gathers atomic propositions and modal subformulas. The Lean defines a finite number of formulas that can be composed. The set of all these compositions represents the exhaustive search universe in which the algorithm is looking for a satisfying tree. A tree node corresponds to a valuation of the Lean formulas.

The second stage of the algorithm consists in a least fixpoint computation that builds every relevant binary tree in a bottom-up manner. At the first step of this stage, all possible leaves are considered. At each further step, the algorithm considers every possible parent node that can be connected with a node of the

previous steps. At each step, built subtrees are checked for consistency: for instance if a formula at a node n involve a forward modality $\langle\triangledown\rangle\phi'$, then ϕ' must be verified at the first child of n. Reciprocally, due to converse modalities, a given node may impose restrictions on its possible parent nodes. The algorithm only considers consistent nodes at each step, meaning that the whole subtree of a given node added at a given step provably satisfies a subformula, except its potential top-level backward modalities that will be taken into account at the next step. At each step, counting formulas are verified. Finally, the algorithm terminates whenever:

- either a tree that satisfies the initial formula has been found, and its root does not contain any pending (unproven) backward modality; or
- no more parent nodes can be considered (the exploration of the whole search universe is complete): the formula is unsatisfiable.

The algorithm is proven sound and complete: ϕ is satisfiable if and only if a tree in which ϕ is satisfied at some node is built. Thus either such a tree is built, or ϕ is not satisfiable.

3.2 Preliminaries

We first annotate every counting formula with a fresh *counting proposition* c, written $\langle\alpha\rangle^c_{\#k}\phi$. We first formally define the notions of Lean and nodes. To this end, we first need to extract navigating formulas from counting formulas.

$$
\begin{aligned}
nav(x) &= x\\
nav(p) &= p\\
nav(\top) &= \top\\
nav(c) &= c\\
nav(\neg p) &= \neg p\\
nav(\neg\langle m\rangle\top) &= \neg\langle m\rangle\top\\
nav(\phi_1 \wedge \phi_2) &= nav(\phi_1) \wedge nav(\phi_2)\\
nav(\phi_1 \vee \phi_2) &= nav(\phi_1) \vee nav(\phi_2)\\
nav(\langle m\rangle\phi) &= \langle m\rangle nav(\phi)\\
nav(\mu x.\psi) &= \mu x.nav(\psi)\\
nav(\langle\alpha\rangle^c_{>k}\psi) &= nav((\alpha), \psi \wedge c)\\
nav(\langle\alpha\rangle^c_{\leq k}\psi) &= nav((\alpha), (\psi \wedge c) \vee (\neg\psi \wedge \neg c))\\
nav((\epsilon), \psi) &= \psi\\
nav((m), \psi) &= \langle m\rangle\psi\\
nav((\alpha_1, \alpha_2), \psi) &= nav((\alpha_1), nav((\alpha_2), \psi))\\
nav((\alpha_1 \mid \alpha_2), \psi) &= nav((\alpha_1), \psi) \vee nav((\alpha_2), \psi)\\
nav((\alpha^\star), \psi) &= \mu x.nav(\psi) \vee nav((\alpha), x)
\end{aligned}
$$

We define the *Fisher-Ladner* relation among formulas as follow, where $i = 1, 2$.

$$R^{fl}(\phi_1 \wedge \phi_2, \phi_i), \qquad R^{fl}(\phi_1 \vee \phi_2, \phi_i),$$
$$R^{fl}(\mu x.\phi, \phi[{}^{\mu x.\phi}/_x]), \qquad R^{fl}(\langle\alpha\rangle^c_{\#k}\psi, nav(\langle\alpha\rangle^c_{\#k}\psi)),$$
$$R^{fl}(\langle m\rangle\phi, \phi).$$

The *Fisher-Ladner* closure of a formula ϕ, written $FL(\phi)$, is the set defined as follow.

$$\begin{array}{lcl} FL(\phi)_0 & = & \{\phi\}, \\ FL(\phi)_{i+1} & = & FL(\phi)_i \cup \{\phi' \mid R^{fl}(\phi'', \phi'), \phi'' \in FL(\phi)_i\}, \\ FL(\phi) & = & FL(\phi)_k, \end{array}$$

where k is the smallest integer s.t. $FL(\phi)_k = FL(\phi)_{k+1}$. Note that this set is finite: fixpoints are only expanded once.

The *Lean set of a formula* ϕ includes navigating formulas of the form $\langle m\rangle\top$, every navigating formulas of the form $\langle m\rangle\phi'$ from the Fisher-Ladner closure, every proposition occurring in ϕ, written P_ϕ, every counting proposition, written C, and an extra proposition that does not occur in ϕ used to represent other names, written $p_{\overline{\phi}}$.

$$Lean(\phi) = \{\langle m\rangle\top\} \cup \{\langle m\rangle\phi' \in FL(\phi)\} \cup P_\phi \cup C \cup \{p_{\overline{\phi}}\}$$

A *ϕ–node* , written n^ϕ, is a non-empty subset of $Lean(\phi)$, such that:

- exactly one proposition from $P_\phi \cup \{p_{\overline{\phi}}\}$ is in each ϕ–node;
- when $\langle m\rangle\phi' \in n^\phi$, then $\langle m\rangle\top \in n^\phi$; and
- both $\langle\triangle\rangle\top$ and $\langle\triangleleft\rangle\top$ cannot be in the same ϕ–node.

The set of ϕ–nodes is defined as N^ϕ.

Intuitively, the formula corresponding to a node n^ϕ is the following.

$$n^\phi = \bigwedge_{\psi \in n^\phi} \psi \wedge \bigwedge_{\psi \in Lean(\phi) \setminus n^\phi} \neg\psi$$

When the formula ϕ under consideration is fixed, we often omit the superscript.

A *ϕtree* is either the empty tree $\varnothing$, or a triple $(n^\phi, \Gamma_1, \Gamma_2)$ where Γ_1 and Γ_2 are ϕtrees.

We now turn to the definition of consistency of a ϕtree. First, we define an entailment relation between a node and a formula in Figure 5.

We can now define the consistency relation between nodes of a ϕtree.

$$\frac{}{n \vdash^{\phi} \top} \qquad \frac{\psi \in n}{n \vdash^{\phi} \psi} \qquad \frac{\psi \notin n}{n \vdash^{\phi} \neg\psi} \qquad \frac{n \vdash^{\phi} \psi_1 \qquad n \vdash^{\phi} \psi_2}{n \vdash^{\phi} \psi_1 \wedge \psi_2} \qquad \frac{n \vdash^{\phi} \psi_1}{n \vdash^{\phi} \psi_1 \vee \psi_2}$$

$$\frac{n \vdash^{\phi} \psi_2}{n \vdash^{\phi} \psi_1 \vee \psi_2} \qquad \frac{n \vdash^{\phi} \psi\{{}^{\mu x.\psi}/_{x}\}}{n \vdash^{\phi} \mu x.\psi}$$

Figure 5: Local entailment relation: between nodes and formulas

Two nodes n_1 and n_2 are consistent under modality $m \in \{\triangledown, \triangleright\}$, written $R^{\phi}(n_1, m) = n_2$, iff

$$\forall \langle m \rangle \psi \in Lean(\phi), \langle m \rangle \psi \in n_1 \iff n_2 \vdash^{\phi} \psi$$
$$\forall \langle \overline{m} \rangle \psi \in Lean(\phi), \langle \overline{m} \rangle \psi \in n_2 \iff n_1 \vdash^{\phi} \psi$$

Consistency is checked each time a node is added to the tree, ensuring that forward modalities of the node are indeed satisfied by the nodes below, and that pending backward modalities of the node below are consistent with the added node. Note that do not check counting formulas at this point, as they are globally verified in the next step.

Upon generation of a finished tree, i.e., a tree with no pending backward modality, one may check whether a node of this tree satisfies ϕ. To this end, we first define forward navigation in a ϕtree Γ. Given a path consisting of forward modalities ρ, $\Gamma(\rho)$ is the node at that path. It is undefined if there is no such node.

$$(n, \Gamma_1, \Gamma_2)(\epsilon) = n$$
$$(n, \Gamma_1, \Gamma_2)(\triangledown \rho) = \Gamma_1(\rho)$$
$$(n, \Gamma_1, \Gamma_2)(\triangleright \rho) = \Gamma_2(\rho)$$

We also allow extending the path with backward modalities if they match the last modality of the path.

$$(n, \Gamma_1, \Gamma_2)(\rho \triangledown \triangle) = (n, \Gamma_1, \Gamma_2)(\rho)$$
$$(n, \Gamma_1, \Gamma_2)(\rho \triangleright \triangleleft) = (n, \Gamma_1, \Gamma_2)(\rho)$$

Now, we are able to define an entailment relation along paths in ϕtrees in Figure 6. This relation extends local entailment relation (Figure 5) with checks for counting formulas. Note that the case for fixpoints is contained in the case for formulas with no counting subformula. Note also that $\neg\psi$ in the "less than" case denotes the negation normal form.

We conclude these preliminaries by introducing some final notations. The *root* of a ϕtree is defined as follows.

$$root(\varnothing) = \varnothing$$
$$root((n, \Gamma_1, \Gamma_2)) = n$$

$$\frac{\phi' \text{ does not contain counting formulas} \qquad \Gamma(\rho) \vdash^{\phi} \phi'}{\rho \vdash^{\phi}_{\Gamma} \phi'} \qquad \frac{\rho \vdash^{\phi}_{\Gamma} \phi_1 \qquad \rho \vdash^{\phi}_{\Gamma} \phi_2}{\rho \vdash^{\phi}_{\Gamma} \phi_1 \wedge \phi_2}$$

$$\frac{\rho \vdash^{\phi}_{\Gamma} \phi_1}{\rho \vdash^{\phi}_{\Gamma} \phi_1 \vee \phi_2} \qquad \frac{\rho \vdash^{\phi}_{\Gamma} \phi_2}{\rho \vdash^{\phi}_{\Gamma} \phi_1 \vee \phi_2} \qquad \frac{\rho m \vdash^{\phi}_{\Gamma} \phi'}{\rho \vdash^{\phi}_{\Gamma} \langle m \rangle \phi'}$$

$$\frac{|\{n',\ \rho' \in \alpha \wedge \Gamma(\rho\rho') = n' \wedge n' \vdash^{\phi} \psi \wedge c\}| > k}{\rho \vdash^{\phi}_{\Gamma} \langle \alpha \rangle^{c}_{>k} \psi}$$

$$\frac{\begin{array}{c}|\{n',\ \rho' \in \alpha \wedge \Gamma(\rho\rho') = n' \wedge n' \vdash^{\phi} \psi \wedge c\}| \leq k \\ \forall \rho' \in \alpha, \Gamma(\rho\rho') \vdash^{\phi} (\psi \wedge c) \vee (\neg\psi \wedge \neg c)\end{array}}{\rho \vdash^{\phi}_{\Gamma} \langle \alpha \rangle^{c}_{\leq k} \psi}$$

Figure 6: Global entailment relation (incl. counting formulas)

We extend this notion to multiset of trees and write $root(ST)$ for the multiset of roots of the trees of ST.

The multiset of nodes of a tree is defined as follows.

$$nodes(\varnothing) = \varnothing$$
$$nodes((n, \Gamma_1, \Gamma_2)) = \{n\} \cup nodes(\Gamma_1) \cup nodes(\Gamma_2)$$

We also extend this notion to multiset of trees.

A ϕtree Γ *satisfies* a formula ϕ, written $\Gamma \vdash \phi$, if neither $\langle \triangle \rangle \top$ nor $\langle \triangleleft \rangle \top$ occur in $root(\Gamma)$, and if there is a path ρ such that $\Gamma(\rho) = n$ and $n \vdash^{\phi}_{\Gamma,\rho} \phi$.

A multiset of trees ST *satisfies* a formula ϕ, written $ST \vdash \phi$, when there is a syntactic tree $\Gamma \in ST$ such that $\Gamma \vdash \phi$.

3.3 The Algorithm

We are now ready to present the algorithm, which is parameterized by $K(\phi)$, the maximum number of occurrences of a given node in a path from the root of the tree to a leaf. It builds consistent candidate trees from the bottom up, and checks at each step if one of the built tree satisfies the formula, returning 1 if it is the case. As the set of nodes from which to build the trees is finite, it eventually stops and returns 0 if no satisfying tree has been found.

Algorithm 1 Check Satisfiability of ϕ

$ST \leftarrow \varnothing$
repeat
  $AUX \leftarrow \{(n, \Gamma_1, \Gamma_2) \mid$ {we extend the trees}
    $\mathtt{nmax}(n, \Gamma_1, \Gamma_2) \leq K(\phi) + 2$ {with an available node}
    for i in $\triangledown, \triangleright$ {and each child is either}
    $\Gamma_i = \varnothing$ and $\langle i \rangle \top \notin n$ {an empty tree}
    or $\Gamma_i \in ST$ {or a previously built tree}
      $\langle \bar{i} \rangle \top \in root(\Gamma_i)$ {with pending backward modalities}
      $R^{\phi}(n, i) = root(\Gamma_i)\}$ {checking consistency}
  if $AUX \subseteq ST$ **then**
    return 0 {No new tree was built}
  end if
  $ST \leftarrow ST \cup AUX$
until $ST \vdash \phi$
return 1

$$
\begin{aligned}
&K(p) = K(\neg p) = K(\neg\langle m \rangle \top) = K(\top) = K(x) = 0\\
&K(\phi_1 \wedge \phi_2) = K(\phi_1 \vee \phi_2) = K(\phi_1) + K(\phi_2)\\
&K(\langle m \rangle \phi) = K(\mu x.\phi) = K(\phi)\\
&K(\langle \alpha \rangle_{\#k} \psi) = k + 1
\end{aligned}
$$

Figure 7: Occurrences bound

We now define the auxiliary `nmax` function as follows, where `max` is the usual maximum function between integers.

$$
\begin{aligned}
\mathtt{nmax}(n, \Gamma_1, \Gamma_2) &= \mathtt{max}(\mathtt{nmax}(n, \Gamma_1), \mathtt{nmax}(n, \Gamma_2))\\
\mathtt{nmax}(n, (n, \Gamma_1, \Gamma_2)) &= 1 + \mathtt{nmax}(\Gamma_1, \Gamma_2)\\
\mathtt{nmax}(n, (n', \Gamma_1, \Gamma_2)) &= \mathtt{nmax}(\Gamma_1, \Gamma_2) \quad \text{if } n \neq n'\\
\mathtt{nmax}(n, \varnothing) &= 0
\end{aligned}
$$

Note a formula $\mu x.\phi$ can be rewritten in an equivalent formula such that x in ϕ is only present in formulas with the form $\langle m \rangle x$. With this last observation, we now define the parameter for the number of occurrence of the same node in the tree in Figure 7.

Consider for instance the formula $\phi = p_1 \wedge \langle \triangledown \rangle \langle \triangleright^{\star} \rangle_{>1} p_2$. The computed Lean is as follows, where $\psi = \mu x.p_2 \vee \langle \triangleright \rangle x$.

$$\{p_1, p_2, p_3, \langle \triangledown \rangle \top, \langle \triangleright \rangle \top, \langle \triangle \rangle \top, \langle \triangleleft \rangle \top, \langle \triangledown \rangle \psi, \langle \triangleright \rangle \psi\}$$

Proposition p_3 represents names other than p_1 and p_2. We now compute the bound on nodes: $K = 2$.

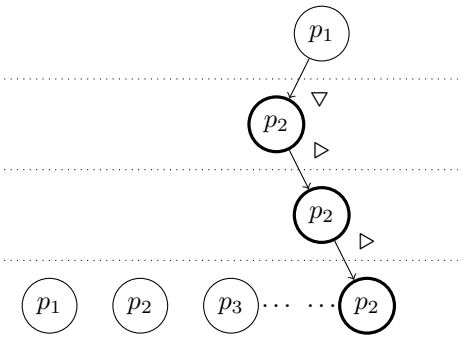

Figure 8: Checking $\phi = p_1 \wedge \langle\triangledown\rangle\langle\triangleright^\star\rangle_{>2} p_2$

After the first step, ST consists of the trees of the form $(\{p_i\}, \varnothing, \varnothing)$ and $(\{p_i, \langle\overline{j}\rangle\top\}, \varnothing, \varnothing)$, with $i \in \{1,2,3\}$ and $j \in \{\triangledown, \triangleright\}$. At this point the three finished trees in ST are tested and found not to satisfy ϕ.

After the second iteration many trees are created, but the one of interest is the following.

$$T_0 = (\{p_2, \langle\triangleright\rangle\top, \langle\triangle\rangle\top, \langle\triangleright\rangle\psi\}, \varnothing, (\{p_2, \langle\triangleleft\rangle\top\}, \varnothing, \varnothing))$$

The third iteration yields the tree $(\{p_1, \langle\triangledown\rangle\psi, \langle\triangledown\rangle\top\}, T_0, \varnothing)$, which is found to satisfy ϕ at path ϵ. As the nodes at every step are different, the limit is not reached. Figure 8 depicts a graphical representation of the example where counted nodes are drawn as thick circles.

3.4 Termination

Proving termination of the algorithm is straightforward, as only a finite number of trees may be built and the algorithm stops as soon as it cannot build a new tree.

3.5 Soundness

If the algorithm terminates with a candidate, we show that the initial formula is satisfiable. Let Γ, ρ the ϕtree and path such that $\rho \vdash^{\phi}_{\Gamma} \phi$. We extract a tree from Γ and show that the interpretation of ϕ for this tree includes the node at path ρ.

We write $T(\Gamma)$ for the tree (N, R, L) defined as follows. We first rewrite Γ such that each node n is replaced by the path to reach it.

$$\begin{aligned} path(n, \Gamma_1, \Gamma_2) &\to (\epsilon, path(\triangledown, \Gamma_1), path(\triangleright, \Gamma_2)) \\ path(\rho, (n, \Gamma_1, \Gamma_2)) &\to (\rho, path(\rho\triangledown, \Gamma_1), path(\rho\triangleright, \Gamma_2)) \\ path(\rho, \varnothing) &\to \varnothing \end{aligned}$$

We then define:

- $N = nodes(path(\Gamma))$;
- for every $(\rho, \Gamma_1, \Gamma_2)$ in $path(\Gamma)$ and $i = \triangledown, \triangleright$, if $\Gamma_i \neq \varnothing$ then $R(\rho, i) = \rho i$ and $R(\rho i, \bar{i}) = \rho$; and
- for all $\rho \in N$ if $p \in \Gamma(\rho)$ then $L(\rho) = p$.

Lemma 3.1. *Let ψ a subformula of ϕ with no counting formula. If $\Gamma(\rho) \vdash^{\phi} \psi$ then we have $\rho \in [\![\psi]\!]^{T(\Gamma)}_{\varnothing}$.*

Proof. We proceed by induction on the lexical ordering of the number of unfolding of ψ that are required for $T(\Gamma)$, and of the size of the formula.

The base cases are $\top$, atomic or counting propositions, and negated forms. These are immediate by definition of $[\![\psi]\!]^{T(\Gamma)}_{\varnothing}$. The cases for disjunction and conjunction are immediate by induction (the formula is smaller). The case for fixpoints is also immediate by induction, as the number of unfoldings required decreases, and as $[\![\mu x.\psi]\!]^{T(\Gamma)}_{\varnothing} = [\![\psi\{^{\mu x.\psi}/_{x}\}]\!]^{T(\Gamma)}_{\varnothing}$.

The last case is the presence of a modality $\langle m \rangle \psi$ from the ϕnode $\Gamma(\rho)$. In this case we rely on the fact that the nodes $\Gamma(\rho m)$ and $\Gamma(\rho)$ are consistent to derive $\rho m \vdash^{\phi} \psi$. We then conclude by induction. □

Theorem 3.2 (Soundness)**.** *If $\rho \vdash^{\phi}_{\Gamma} \phi$ then $\rho \in [\![\phi]\!]^{T(\Gamma)}_{\varnothing}$*

Proof. We proceed by induction on the derivation of $\rho \vdash^{\phi}_{\Gamma} \phi$.

The proof is a consequence of the more general result $\rho' \vdash^{\phi}_{\Gamma} \phi' \implies \rho' \in [\![\phi']\!]^{T(\Gamma)}_{\varnothing}$ for any subformula of ϕ', by induction on the derivation of $\Gamma(\rho') \vdash^{\phi}_{\Gamma,\rho} \phi'$. If ϕ' has no counting formula, the result is immediate by Lemma 3.1. Most cases are immediate by induction. As concerns the case for counting formulas, each hypothesis $n' \vdash^{\phi} \psi \wedge c$ has as hypothesis $n' \vdash^{\phi} \psi$. This is enough to conclude by induction for the "greater than" case. For the "less than" case, every node that is not counted has to satisfy $\neg\psi \wedge \neg c$, so in particular $\neg\psi$, and we conclude by induction. □

3.6 Completeness

Our proof proceeds in two step. We build a ϕtree that satisfies the formula, then we proceed to show it is actually built by the algorithm.

Assume that formula ϕ is satisfiable by a tree T. We consider the smallest such tree (i.e., the tree with the fewest number of nodes) and fix $n^{\star}$, a node witnessing satisfiability.

We now build a ϕtree homomorphic to T, called the Lean labeled version of ϕ, written $\Gamma(T, \phi)$, and defined as follows.

First, we annotate counted nodes along with their corresponding counting proposition, yielding a new tree T_c. Starting from $n^{\star}$ and by induction on ϕ, we proceed as follows. For formulas with no counting subformula, including recursion, we stop. For conjunction and disjunction of formulas, we recursively annotate according to both subformulas. For modalities, recursively annotate

from the node under the modality. For $\langle\alpha\rangle^c_{\leq k}\psi$, we annotate every selected node with the counting proposition corresponding to the formula. For $\langle\alpha\rangle^c_{>k}\psi$, we annotate exactly $k+1$ selected nodes.

We now extend the semantics of formulas to take into account counting propositions and annotated nodes, written $[\![\cdot]\!]^{T_c}_V$. The definition is identical to Figure 4, with one addition and two changes. The addition is for counting propositions, which we define as $n \in [\![c]\!]^{T_c}_V$ iff n is annotated by c. The two changes are for counting propositions, which we define as follows, selected only nodes that are annotated.

$$[\![\langle\alpha\rangle_{\leq k}\phi']\!]^{T_c}_V = \{n, |\{n' \in [\![\phi']\!]^{T_c}_V \cap [\![c]\!]^{T_c}_V, n \xrightarrow{\alpha} n'\}| \leq k\}$$
$$[\![\langle\alpha\rangle_{>k}\phi']\!]^{T_c}_V = \{n, |\{n' \in [\![\phi']\!]^{T_c}_V \cap [\![c]\!]^{T_c}_V, n \xrightarrow{\alpha} n'\}| > k\}$$

We show that this modification of the semantics does no change the satisfiability of the formula.

Lemma 3.3. *We have $n^\star \in [\![\phi]\!]^{T_c}_\varnothing$.*

Proof. We proceed by recursion on the derivation $n^\star \in [\![\phi]\!]^T_\varnothing$. The cases where no counting formula is involved, thus including fixpoints, are immediate, as the selected nodes are identical. The disjunction, conjunction, and modality cases are also immediate by induction. The interesting cases are the counting formulas.

For $\langle\alpha\rangle^c_{>k}\psi$, as there are exactly $k+1$ nodes annotated, the property is true by induction. For $\langle\alpha\rangle^c_{\leq k}\psi$, we rely on the fact that every counted node is annotated. We conclude by remarking that ψ does not contain a counting formula, thus we have $[\![\psi]\!]^{T_c}_V = [\![\psi]\!]^T_V$ and $[\![\neg\psi]\!]^{T_c}_V = [\![\neg\psi]\!]^T_V$. □

To every node n, we associate n^ϕ, a subset of formulas of the Lean selecting the node.

$$n^\phi = \{\phi_0 \mid n \in [\![\phi_0]\!]^T_\varnothing, \phi_0 \in \mathit{Lean}(\phi)\}$$

Note that this is a ϕ-node as it contains one and exactly one proposition, and if it includes a modal formula $\langle m\rangle\psi$, then it also includes $\langle m\rangle\top$.

The tree $\Gamma(T, \phi)$ is then built homomorphically to T.

In the remainder of this section, we write Γ for $\Gamma(T, \phi)$. We first check that Γ is consistent, starting with local consistency.

In the following, we say a formula ψ is induced by the lean of ϕ, written $\psi \dot{\in} \mathit{Lean}(\phi)$, if it consists of the conjunction and disjunction of formulas from the lean as defined in Figure 9.

Lemma 3.4. *Let $\langle m\rangle\psi$ be a formula in $\mathit{Lean}(\phi)$, and let ψ' be ψ after unfolding its fixpoint formulas not under modalities. We have $\psi' \dot{\in} \mathit{Lean}(\phi)$.*

Proof. By definition of the lean and of the $\dot{\in}$ relation. □

Lemma 3.5. *Let ψ be a formula induced by $\mathit{Lean}(\phi)$. We have $n \in [\![\psi]\!]^{T_c}_\varnothing$ if and only if $n^\phi \vdash^\phi \psi$.*

$$\frac{\psi \in Lean(\phi)}{\psi \dot{\in} Lean(\phi)} \qquad \frac{\psi_1 \dot{\in} Lean(\phi) \qquad \psi_2 \dot{\in} Lean(\phi)}{\psi_1 \wedge \psi_2 \dot{\in} Lean(\phi)}$$

$$\frac{\psi_1 \dot{\in} Lean(\phi) \qquad \psi_2 \dot{\in} Lean(\phi)}{\psi_1 \vee \psi_2 \dot{\in} Lean(\phi)} \qquad \frac{}{\top \dot{\in} Lean(\phi)} \qquad \frac{\psi \in (P_\phi \cup \langle m \rangle \top \cup C)}{\neg\psi \dot{\in} Lean(\phi)}$$

Figure 9: Formula induced by a lean

Proof. We proceed by induction on ψ. The base cases (the formula is in the ϕ-node or is a negation of a lean formula not in the ϕ-node) hold by definition of n^ϕ. The inductive cases are straightforward as these formulas only contain fixpoints under modalities. □

Lemma 3.6. *Let n_1 and n_2 such that $R(n_1, m) = n_2$ with $m \in \{\triangledown, \triangleright\}$. We have $R^\phi(n_1^\phi, m) = n_2^\phi$.*

Proof. Let $\langle m \rangle \psi$ be a formula in $Lean(\phi)$. We show that $\langle m \rangle \psi \in n_1^\phi \iff n_2^\phi \vdash^\phi \psi$. We have $\langle m \rangle \psi \in n_1^\phi$ if and only if $n_1 \in [\![\langle m \rangle \psi]\!]_\varnothing^{T_c}$ by definition of n_1^ϕ, which in turn holds if and only if $n_2 = R(n_1, m) \in [\![\psi]\!]_\varnothing^{T_c}$. We now consider ψ' which is ψ after unfolding its fixpoint formulas not under modalities. We have $[\![\psi']\!]_\varnothing^{T_c} = [\![\psi]\!]_\varnothing^{T_c}$ and we conclude by Lemmas 3.4 and 3.5. □

We now turn to global consistency, taking counting formulas into account.

Lemma 3.7. *Let ϕ_s be a subformula of ϕ, and ρ be a path from the root in T such that $T(\rho) \in [\![\phi_s]\!]_\varnothing^{T_c}$. We then have $\rho \vdash_\Gamma^\phi \phi_s$.*

Proof. We proceed by induction on ϕ_s.

If ϕ_s does not contain any counting formula, we consider ϕ_s' which is ϕ_s ofter unfolding its fixpoint formulas not under modalities. We have $[\![\phi_s']\!]_\varnothing^{T_c} = [\![\phi_s]\!]_\varnothing^{T_c}$ and $\phi_s' \dot{\in} Lean(\phi)$. We conclude by Lemma 3.5.

For most inductive cases, the proof is immediate by induction, as the formula size decreases.

For $\langle \alpha \rangle_{>k}^c \psi$, we have by induction form every counted node $\Gamma(\rho\rho') \vdash^\phi \psi$ and $\Gamma(\rho\rho') \vdash^\phi c$. We conclude by the conjunction rule and by the counting rule of Figure 6.

For $\langle \alpha \rangle_{\leq k}^c \psi$, we proceed as above for the counted nodes. For the nodes that are not counted, have $[\![\neg\psi]\!]_V^{T_c} = [\![\neg\psi]\!]_V^T$ and by soundness, we have $\Gamma(\rho\rho') \vdash^\phi \neg\psi$. We conclude by remarking that the node is not annotated by c, hence $\Gamma(\rho\rho') \vdash^\phi \neg c$. □

We now need to show that the ϕtree Γ is actually built by the algorithm. The proof that it is the case follows closely the one from [GLS07], with a crucial exception: we need to make sure there are enough instances of each formula.

Indeed, in [GLS07], the algorithm uses a ϕtype (a subset of $Lean(\phi)$) at most once on each branch from the root to a leaf of the built tree. This yields a simple condition to stop the algorithm and conclude the formula is unsatisfiable. However, in the presence of counting formulas, a given ϕtype may occur more than once on a branch. To maintain the termination of the algorithm, we bound the number of identical ϕtype that may be needed by $K(\phi)$ as defined in Figure 7. We now check that this bound is sufficient to build a tree for any satisfiable formula.

We recall that ϕ is a satisfiable formula and T is a smallest tree such that ϕ is satisfied, and $n^\star$ is a witness of satisfiability.

We proceed in two steps: first we show that counted nodes (with counted propositions) imply a bound on the number of identical ϕtypes on a branch for a smallest tree. Second, we show that this minimal marking is bound by $K(\phi)$.

In the following, we call counted nodes and node $n^\star$ *annotations*.

We now define the projection of an annotation on a path. Let ρ be a path from the root of the tree to a leaf. An annotation projects on ρ at ρ_1 if $\rho = \rho_1\rho_2$, the annotation is at $\rho_1\rho_m$, and ρ_2 shares no prefix with ρ_m.

Lemma 3.8. *Let Γ' be the annotated tree, ρ a path from the root of the tree to a leaf, n_1 and n_2 two distinct nodes of ρ such that $n_1^\phi = n_2^\phi$. Then either annotations projects both on ρ at n_1 and n_2, or an annotation projects strictly between n_1 and n_2.*

Proof. We proceed by contradiction: we assume there is no annotation that projects between n_1 and n_2 and at most one of them has an annotation that projects on it. Without loss of generality, we assume that n_2 is below n_1 in the tree.

Assume neither n_1 nor n_2 is annotated (through projection). We show that the tree where $R(n_1, \triangledown) \leftarrow R(n_2, \triangledown)$ and $R(n_1, \triangleright) \leftarrow R(n_2, \triangleright)$ still satisfies ϕ at n, a contradiction since this tree is strictly smaller. Let T_s be this smaller tree, Γ_s the corresponding ϕtree, and for every path ρ of Γ, let ρ_s be the potentially shorter path if it exists (i.e., if it was not removed when pruning the tree). More precisely, let ρ_1 be the path to n_1 and $\rho_1\rho_2$ be the path to n_2. If $\rho' = \rho'_1\rho'_3$ where ρ'_1 is a prefix of ρ_1 and the paths are disjoint from there, then $\Gamma_s(\rho') = \Gamma(\rho')$. If $\rho' = \rho_1\rho_2\rho_3$, then $\Gamma_s(\rho_1\rho_3) = \Gamma(\rho')$.

First, as there was no annotation projected, n is still part of this tree at a path ρ_s. We show that we have $\rho_s \vdash^\phi_{\Gamma_s} \phi$ by induction on the derivation $\rho \vdash^\phi_\Gamma \phi$. Let $\rho' \vdash^\phi_\Gamma \phi'$ in the derivation, assuming that ρ'_s is defined.

The case where ϕ' does not mention any counting formula is trivial: $\Gamma(\rho') = \Gamma_s(\rho'_s)$ thus local entailment is immediate.

Conjunction and disjunction are also immediate by induction.

For the modality case, we first need to prove an additional property. If $\rho' \vdash^\phi_\Gamma \langle m\rangle\phi'$ and ϕ' contains a counting formula, then $\rho'm$ is either a prefix of ρ_1 followed by a disjoint path, or it includes $\rho_1\rho_2$. We prove this property by contradiction. The formula $\langle m\rangle\phi'$ is both in $\Gamma(\rho_1)$ and in $\Gamma(\rho_1\rho_2)$. We consider the outermost counting formula in ϕ' which we write ϕ'_c. It presence

implies the occurrence of a counting proposition c in the formula. Since counting propositions are distinct for distinct syntactic occurrences of a formula, this implies that the corresponding counting proposition is either under a fixpoint (which is impossible), or under an enclosing counting formula, which is also impossible. We thus have a contradiction

We now turn to the counting case $\langle\alpha\rangle^c_{\#k}\psi$. We say that a path *does not cross over* when this path does not contain n_1 nor n_2. For nodes that are reached using paths that do not cross over, we conclude by induction that they are also counted. We now show that the remaining nodes for which a crossover happened are also reached. Without loss of generality, assume that ρ' is a prefix of ρ_1 (the counting formula is in the "top" part of the tree), and let ρ_n be the path from the counting formula to the counted node (ρ_n is an instance of the trail α). This path is of the shape $\rho'_1\rho_2\rho_c$, with $\rho_1 = \rho'\rho'_1$. We now show that the path $\rho'_1\rho_c$ is an instance of α if and only if ρ_n is, thus the same node is still counted.

Recall that α is of the shape $\alpha_1, \ldots, \alpha_n, \alpha_{n+1}$ where α_1 to α_n are of the form $\alpha_r^\star$ and where α_{n+1} does not contain a repeated trail. We say that a prefix ρ_p of a path ρ *stops at* i if there is a suffix ρ_s such that $\rho_p\rho_s$ is still a prefix of ρ, if $\rho_p\rho_s \in \alpha_1, \ldots, \alpha_i$, and if there is no shorter suffix ρ'_s and j such that $\rho_p\rho'_s \in \alpha_1, \ldots, \alpha_j$. (Intuitively, α_i is the trail being used when matching the end of ρ_p.) Note that i may not be unique as a path may be matched in different ways by a trail. We now show that there are $i \leq j \leq n$ such that both ρ'_1 stops at i and $\rho'_1\rho_2$ stop at j. We thus show that j cannot be $n+1$. Recall that α_{n+1} does not contain a repeated subtrail. If n_2^ϕ does not contain the counted proposition c (which may happen in the case of a "less than" counting where the target is not counted), then neither does n_1^ϕ, which is a contradiction to the fact that $\alpha i, \ldots, \alpha_{n+1}$ is not empty (in that case the counted proposition is necessarily mentioned). Thus n_2^ϕ contain formulas without a fixpoint (as the trail is not repeated) mentioning c. Consider the largest such formula. By an induction on the path ρ_2, we build a strictly larger formula that occurs in n_1^ϕ. This a contradiction to the hypothesis that $n_1^\phi = n_2^\phi$.

We now consider the suffixes ρ_s^1 and ρ_s^2 computed when stating that the paths stop at i and j. These suffixes correspond to the path matching the end of α_i and α_j, respectively (before the next iteration or switching to the next formula). They have matching formulas in n_1^ϕ and n_2^ϕ. As the formulas are present in both nodes, then the remainder of the paths ($\rho_2\rho_c$ and ρ_c) are instances of $(\rho_s^1|\rho_s^2)\alpha_i \ldots \alpha_{n+1}$, thus $\rho'_1\rho_c$ is an instance of α if and only if ρ_n is.

In the case of "greater than" counting, we conclude immediately by induction as the same nodes are selected (thus there are enough). In the case of "less than", we need to check that no new node is counted in the smaller tree. Assume it is not the case for the formula $\langle\alpha\rangle_{\leq k}\psi$, thus there is a path $\rho_n \in \alpha$ to a node satisfying ψ. As the same node can be reached in Γ, and as we have $\Gamma(\rho'\rho_n) \vdash^\phi \neg\psi$ by induction, we have a contradiction.

This concludes the proof when neither n_1 nor n_2 is annotated. The proof is identical when n_2 is annotated. If n_1 is annotated, we look at the first

modality between n_1 and n_2. If it is a $\triangledown$, then we build the smaller tree by doing $R(n_1, \triangledown) \leftarrow R(n_2, \triangledown)$ (we remove the $\triangleright$ subtree from n_2 instead of n_1). Symmetrically, if the first modality is a $\triangleright$, we consider $R(n_1, \triangleright) \leftarrow R(n_2, \triangleright)$ as smaller tree. The rest of the proof proceeds as above. □

Theorem 3.9 (Completeness)**.** *If ϕ is satisfiable, then a satisfying tree is built.*

Proof. The proof proceeds as in [GLS07], we only need to check there are enough copies of each node to build every path. Let ρ be a path from the root of the tree to the leaves. By Lemma 3.8, there are at most $n+1$ identical nodes in this path, where n is the number of marks. The number of marks is $c+1$ where c is the number of counted nodes. We show by an immediate induction on the formula ϕ that c is bound by $K(\phi)$ as defined in Figure 7. We conclude by remarking that $K(\phi)+2$ is the number of identical nodes we allow in the algorithm. □

3.7 Complexity

We now show that the complexity of the satisfiability algorithm is exponential time w.r.t. the formula size. This is achieved in two steps: we first show that the Lean size is linear w.r.t. the formula size, then we show that the algorithm has a single exponential complexity w.r.t. to the Lean size.

Lemma 3.10. *The Lean size is linear in terms of the original formula size.*

Proof Sketch. It was shown in [GLS07] that the Lean size of non counting formulas is linear with respect to the formula size.

We now describe the case for counting formulas. Note that each counting formula introduces only one new counting proposition in the Lean. A first duplication of formulas is considered in the construction of the Lean for "less than" counting formulas. Both, the formula witnessing the counted nodes and its negation are considered. Furthermore, another duplication is introduced for counting formulas of the form $\langle \alpha_1 | \alpha_2 \rangle_{\#k} \phi$. Each of these duplications only doubles the size of the Lean. Hence, the Lean size remains linear w.r.t to the original formula size. □

Theorem 3.11. *The satisfiability algorithm for the logic is decidable in time $2^{O(n)}$, where n is the Lean size.*

Proof Sketch. The cardinality of nodes set is 2^n. The number of occurrences of each node in the tree is bounded by $K(\phi) \leq k * m$, where k is the greatest constant occurring in the counting formulas and m is the number of counting subformulas. Hence the number of steps in the algorithm is bounded by $2^n * k * m$.

As for the functions at each step, `nmax` is a single traversal to the tree. Since the entailment relation involved in the definition of R^ϕ is only local, R^ϕ is performed in linear time.

The number of choices to form trees (triples) at each step is restricted by $3 * (2^n * k * m)$.

The global entailment relation involves four exponential time traversals: the number of trees, the number of nodes at each tree, the number of traversals for the entailment relation of counting formulas, and the cost of each of such traversals. Hence it takes no more than $4 * (2^n * k)$ time.

□

Theorem 3.11 states the complexity for the logic defined in Figure 2. We now state that the same complexity upper-bound holds if we additionally consider counting formulas of the form $\langle\triangledown\rangle\langle\triangleright^*\rangle_{\#k}\phi$ in the scope of a fixpoint operator (as presented in Section 2.7).

Lemma 3.12. *Given a formula ϕ where counting subformulas ψ only count children nodes, if every counting subformula ψ is replaced by the equivalent fixpoint formula $ch(\psi)$ in ϕ, $\phi[^{ch(\psi)}/_{\psi}]$, then $Lean(\phi[^{ch(\psi)}/_{\psi}]) \leq Lean(\phi) * k^l$, where k is greatest numerical constraint of the counting subformulas, and l is the greatest level nesting of counting subformulas.*

Proof Sketch. It is proven by induction on the structure of ϕ, and in the case of counting formulas, another induction is done on the numerical constraint. □

Corollary 3.13. *The logic supporting counting formulas only on children in the scope of fixpoint formulas or another counting formula is decidable in $2^{O(n*k^l)}$, where k is the greatest cardinality constraint and l is the greatest nesting level of counting formulas.*

Proof. Immediate from Theorem 3.11 and Lemma 3.12. □

4 Application to XML Trees

4.1 XPath Expressions

XPath [CD99] was introduced as part of the W3C XSLT transformation language to have a non-XML format for selecting nodes and computing values from an XML document (see [GLS07] for a formal presentation of XPath). Since then XPath has become part of several other standards, in particular it forms the "navigation subset" of the XQuery language.

In their simplest form XPath expressions look like "directory navigation paths". For example, the XPath

```
/company/personnel/employee
```

navigates from the root of a document through the top-level "company" node to its "personnel" child nodes and on to its "employee" child nodes. The result of the evaluation of the entire expression is the set of all the "employee" nodes that can be reached in this manner. At each step in the navigation, the selected nodes for that step can be filtered with a test. Of special interest to us are the predicates that test node's count or the selected node's position in the previous step's selection. For example, if we ask for

```
/company/personnel/employee[position()=2]
```

then the result is *all* employee nodes that are the *second* employee node among the employee child nodes of each personnel node selected by the previous step.

XPath also makes it possible to combine the capability of searching along "axes" other than the shown "children of" with counting constraints. For example, if we ask for

```
/company[count(descendant::employee<=300)]/name
```

then the result consists of the company names with less than 300 employees in total (the axis "descendant" is the transitive closure of the default – and often omitted – axis "child").

The syntax and semantics of Core XPath expressions are respectively given on Figure 10 and Figure 11. An XPath expression is interpreted as a relation between nodes. The considered XPath fragment allows absolute and relative paths, path union, intersection, composition, as well as node tests and qualifiers with counting operators, conjunction, disjunction, negation, and path navigation. Furthermore, it supports all XPath axes allowing multidirectional navigation.

$$
\begin{aligned}
\text{Axis} &::= \text{self} \mid \text{child} \mid \text{parent} \mid \text{descendant} \mid \text{ancestor} \mid \\
&\quad \text{following-sibling} \mid \text{preceding-sibling} \mid \\
&\quad \text{following} \mid \text{preceding} \\
\text{NameTest} &::= \text{QName} \mid * \\
\text{Step} &::= \text{Axis::NameTest} \\
\text{PathExpr} &::= \text{PathExpr/PathExpr} \mid \text{PathExpr[Qualifier]} \mid \text{Step} \\
\text{Qualifier} &::= \text{PathExpr} \mid \text{CountExpr} \mid \text{not Qualifier} \mid \\
&\quad \text{Qualifier and Qualifier} \mid \text{Qualifier or Qualifier} \mid @n \\
\text{CountExpr} &::= \text{count}(\text{PathExpr}') \text{ Comp } k \\
\text{PathExpr}' &::= \text{PathExpr}'/\text{PathExpr}' \mid \text{PathExpr}'[\text{Qualifier}'] \mid \text{Step} \\
\text{Qualifier}' &::= \text{PathExpr}' \mid \text{not Qualifier}' \mid \text{Qualifier}' \text{ and Qualifier}' \\
&\quad \mid \text{Qualifier}' \text{ or Qualifier}' \mid @n \\
\text{Comp} &::= \leq \mid > \mid \geq \mid < \mid = \\
\text{XPath} &::= \text{PathExpr} \mid /\text{PathExpr} \mid \text{XPath union PathExpr} \mid \\
&\quad \text{XPath intersect PathExpr} \mid \text{XPath except PathExpr}
\end{aligned}
$$

Figure 10: Syntax of Core XPath Expressions.

It was already observed in [GR05, tCM09] that using positional information in paths reduces to counting (at the cost of an exponential blow-up). For example, the expression

```
child::a[position()=5]
```

$$
\begin{aligned}
[\![\text{Axis::NameTest}]\!] &= \{(x,y) \in N^2 \mid x(\text{Axis})y \text{ and } \\
&\quad\ y \text{ satisfies NameTest}\} \\
[\![/\text{PathExpr}]\!] &= \{(r,y) \in [\![\text{PathExpr}]\!] \mid \\
&\quad\ r \text{ is the root}\} \\
[\![P_1/P_2]\!] &= [\![P_1]\!] \circ [\![P_2]\!] \\
[\![P_1 \text{ union } P_2]\!] &= [\![P_1]\!] \cup [\![P_2]\!] \\
[\![P_1 \text{ intersect } P_2]\!] &= [\![P_1]\!] \cap [\![P_2]\!] \\
[\![P_1 \text{ except } P_2]\!] &= [\![P_1]\!] \setminus [\![P_2]\!] \\
[\![\text{PathExpr[Qualifier]}]\!] &= \{(x,y) \in [\![\text{PathExpr}]\!] \mid \\
&\quad\ y \in [\![\text{Qualifier}]\!]_{\text{Qualif}}\}
\end{aligned}
$$

$$
\begin{aligned}
[\![\text{PathExpr}]\!]_{\text{Qualif}} &= \{x \mid \exists y.(x,y) \in [\![\text{PathExpr}]\!]\} \\
[\![\text{count(PathExpr) Comp } k]\!]_{\text{Qualif}} &= \{x \in N \mid \\
&\quad\ |\{y \mid (x,y) \in [\![\text{PathExpr}]\!]\}| \\
&\quad\ \text{satisfies Comp } k\} \\
[\![\text{not } Q]\!]_{\text{Qualif}} &= N \setminus [\![Q]\!]_{\text{Qualif}} \\
[\![Q_1 \text{ and } Q_2]\!]_{\text{Qualif}} &= [\![Q_1]\!]_{\text{Qualif}} \cap [\![Q_1]\!]_{\text{Qualif}} \\
[\![Q_1 \text{ or } Q_2]\!]_{\text{Qualif}} &= [\![Q_2]\!]_{\text{Qualif}} \cup [\![Q_2]\!]_{\text{Qualif}}
\end{aligned}
$$

Figure 11: Semantics of Core XPath Expressions

first selects the "a" nodes occurring as children of the current context node, and then keeps those occurring at the 5th position. This expression can be rewritten into the semantically equivalent expression:

```
child::a[count(preceding-sibling::a)=4]
```

which constraints the number of preceding siblings named "a" to 4, so that the qualifier becomes true only for the 5th child "a". A general translation of positional information in terms of counting operators [GR05, tCM09] is summarized on Figure 12, where $\ll$ denotes the document order (depth-first left-to-right) relation in a tree. Note that translated path expressions can in turn be expressed into the core XPath fragment of Figure 10 (at the cost of another exponential blow-up). Indeed, expressions like $\text{PathExpr}/(\text{PathExpr}_2 \text{ except } \text{PathExpr}_3)/\text{PathExpr}_4$ must be rewritten into expressions where binary connectives for paths occur only at top level, as in:

$$\begin{array}{l}\text{PathExpr}/\text{PathExpr}_2/\text{PathExpr}_4 \text{ except}\\ \text{PathExpr}/\text{PathExpr}_3/\text{PathExpr}_4\end{array}$$

$$\begin{array}{rl}\text{PathExpr}[\text{position}() = 1] \equiv & \text{PathExpr except } (\text{PathExpr}/\ll)\\ \text{PathExpr}[\text{position}() = k+1] \equiv & (\text{PathExpr intersect}\\ & \quad (\text{PathExpr}[k]/\ll))[\text{position}()=1]\\ \ll \equiv & (\text{descendant::*}) \text{ union } (\text{a-o-s::*}/\\ & \quad \text{following-sibling::*}/\text{d-or-s::*})\\ \text{a-or-s::*} \equiv & \text{ancestor::* union self::*}\\ \text{d-or-s::*} \equiv & \text{descendant::* union self::*}\end{array}$$

Figure 12: Positional Information as Syntactic Sugars [GR05, tCM09]

We focus on Core XPath expressions involving the counting operator (see Figure 10). The XPath fragment without the counting operator (the navigational fragment) was already linearly translated into μ-calculus in [GLS07]. The contributions presented in this paper allow to equip this navigational fragment with counting features such as the ones formulated above. Logical formulas capture the aforementioned XPath counting constraints. For example, consider the following XPath expression:

```
child::a[count(descendant::b[parent::c])>5]
```

This expression selects the children nodes named "a" provided they have more than 5 descendants which (1) are named "b" and (2) whose parent is named "c". The logical formula denoting the set of children nodes named "a" is:

$$\psi = a \wedge \langle \triangleleft^*, \triangle \rangle \top$$

The logical translation of the above XPath expression is:

$$\psi \wedge \langle\triangledown\rangle\langle(\triangledown|\triangleright)^\star\rangle_{>5}(b \wedge \mu x.\langle\triangle\rangle c \vee \langle\triangleleft\rangle x)$$

This formula holds for nodes selected by the XPath expression. A correspondence between the main XPath axes over unranked trees and modal formulas over binary trees is given in Figure 13. In this figure, each logical formula holds for nodes selected by the corresponding XPath axis from a context γ.

Path	Logical formula
γ/self::*	γ
γ/child::*	$\langle\triangleleft^*,\triangle\rangle\gamma$
γ/parent::*	$\langle\triangledown\rangle\langle\triangleright^*\rangle\gamma$
γ/descendant::*	$\langle(\triangleleft \mid \triangle)^*,\triangle\rangle\gamma$
γ/ancestor::*	$\langle\triangledown\rangle\langle(\triangledown \mid \triangleright)^*\rangle\gamma$
γ/following-sibling::*	$\langle\triangleleft\rangle\langle\triangleleft^*\rangle\gamma$
γ/preceding-sibling::*	$\langle\triangleright\rangle\langle\triangleright^*\rangle\gamma$

Figure 13: XPath axes as modalities over binary trees.

Let consider another example (XPath expression e_1):

```
child::a/child::b[count(child::e/descendant::h])>3]
```

Starting from a given context in a tree, this XPath expression navigates to children nodes named "a" and selects their children named "b". Finally, it retains only those "b" nodes for which the qualifier between brackets holds. The first path can be translated in the logic as follows:

$$\vartheta = b \wedge \mu x.\langle\triangle\rangle(a \wedge \mu x'.\langle\triangle\rangle\top \vee \langle\triangleleft\rangle x') \vee \langle\triangle\rangle x$$

This example requires a more sophisticated translation in the logic. This is because it makes implicit that "e" nodes (whose existence is simply tested for counting purposes) must be children of selected "b" nodes. The translation of the full aforementioned XPath expression is as follows:

$$\vartheta \wedge @n \wedge \langle(\triangle \mid \triangleleft)^*,(\triangledown \mid \triangleright)^*\rangle_{>3}\eta$$

where $@n$ is a new fresh nominal used to mark a "b" node which is filtered by the qualifier and the formula η describes the counted "h" nodes:

$$\eta = h \wedge \mu x.\langle\triangle\rangle(e \wedge \mu x'.\langle\triangle\rangle @n \vee \langle\triangleleft\rangle x') \vee \langle\triangleleft\rangle x \vee \langle\triangle\rangle x$$

Intuitively, the general idea behind the translation is to first translate the leading path, use a fresh nominal for marking a node which is filtered, then find at least "3" instances of "h" nodes from which we can reach back the marked node via the inverse path of the counting formula.

Since trails make it possible to navigate but not to test properties (like existence of labels), we test for labels in the counted formula η and we use a general navigation $(\triangle \mid \triangleleft)^*, (\triangledown \mid \triangleright)^*$ to look for counted nodes everywhere in the tree. Introducing the nominal is necessary to bind the context properly (without loss of information). Indeed, the XPath expression e_1 makes implicit that a "e" node must be a child of a "b" node selected by the outer path. Using a nominal, we restore this property by connecting the counted nodes to the initial single context node.

Lemma 4.1. *The translation of Core XPath expressions with counting constraints into the logic is linear.*

It is proven by structural induction in a similar manner to [GLS07] (in which the translation is proven for expressions without counting constraints). For counting formulas, the use of nominals and the general (constant-size) counting trail make it possible to avoid duplication of trails so that the translation remains linear.

Corollary 4.2. *The equivalence problem for expressions of the form:*

$$PathExpr[count(PathExpr')\#k]$$

where $\# \in \{\leq, >, =\}$ *and* k *is a constant, is decidable. More specifically, the equivalence problem can be decided in exponential time in terms of the expression size and the highest nesting level of counting formulas.*

4.2 Regular Tree Languages with Cardinality Constraints

Regular tree grammars capture most of the schemas in use today [MLMK05]. The logic can express all regular tree languages (it is easy to prove that regular expression types in the manner of e.g., [HVP05] can be linearly translated into the logic: see [GLS07]).

In practice, schema languages often provide shorthands for expressing cardinality constraints on node occurrences. XML Schema notably offers two attributes *minOccurs* and *maxOccurs* for this purpose. For instance, the following XML schema definition:

```
<xsd:element name="a">
  <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="b" minOccurs="4" maxOccurs="9"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

is a notation that restricts the number of occurrences of "b" nodes to be at least 4 and at most 9, as children of "a" nodes. The goal here is to have a succinct notation for expressing regular languages which could otherwise be

exponentially large if written with usual regular expression operators. The above regular requirement can be translated as the formula:

$$\phi \wedge \langle \triangledown \rangle (\langle \triangleright^\star \rangle_{>3} b \wedge \langle \triangleright^\star \rangle_{\leq 9} b)$$

where ϕ corresponds to the regular tree type $a[b^*]$ as follows:

$$\begin{aligned} \phi = \quad & (a \wedge (\neg\langle \triangledown \rangle \top \vee \langle \triangledown \rangle \psi)) \wedge \neg\langle \triangleright \rangle \top \\ \psi = \quad & \mu x.\, (b \wedge \neg\langle \triangledown \rangle \top \wedge \neg\langle \triangleright \rangle \top) \vee (b \wedge \neg\langle \triangledown \rangle \top \wedge \langle \triangleright \rangle x) \end{aligned}$$

This example only involves counting over children nodes. The logic allows counting through more general trails, and in particular arbitrarily deep trails. Trails corresponding to the XPath axes "preceding, ancestor, following" can be used to constrain the context of a schema. The "descendant" trail can be used to specify additional constraints over the subtree defined by a given schema. For instance, suppose we want to forbid webpages containing nested anchors "a" (whose interpretation makes no sense for web browsers). We can build the logical formula f which is the conjunction of a considered schema for webpages (e.g. XHTML) with the formula a/descendant::a in XPath notation. Nested anchors are forbidden by the considered schema iff f is unsatisfiable.

As another example, suppose we want paragraph nodes ("p" nodes) not to be nested inside more than 3 unordered lists ("ul" nodes), regardless of the schema defining the context. One may check for the unsatisfiability of the following formula:

$$p \wedge \langle (\triangle | \triangleleft)^\star, \triangle \rangle_{>3} ul$$

5 Related Work

Counting over graphs The μ-calculus is a propositional modal logic augmented with least and greatest fixpoint operators [Koz82]. Kupferman, Sattler and Vardi study a μ-calculus with graded modalities where one can express, e.g., that a node has at least n successors satisfying a certain property [KSV02]. The modalities are limited in scope since they only count children of a given node.

The μ-calculus has been recently extended with inverse modalities [Var98], nominals [SV01], and graded modalities [KSV02]. If only two of the above constructs are considered, satisfiability of the enriched calculus is EXPTIME-complete [BLMV06]. However, if all of the above constructs are considered simultaneously, the calculus becomes undecidable [BLMV06]. Hopefully, this undecidability result for the case of graphs does not preclude decidable tree logics combining such features.

Counting over trees The notion of Presburger Automata for trees, combining both regular constraints on the children of nodes and numerical constraints given by Presburger formulas, has independently been introduced by Dal Zilio and Lugiez [DZLM04] and Seidl et al. [SSMH04]. Specifically, Dal Zilio and

Lugiez [DZLM04] propose a modal logic for unordered trees called Sheaves logic. This logic allows to impose certain arithmetical constraints on children nodes but lacks recursion (i.e., fixpoint operators) and inverse navigation. Dal Zilio and Lugiez consider the satisfiability and the membership problems. Demri and Lugiez [DL06] showed by means of an automata-free decision procedure that this logic is only PSPACE-complete. Restrictions like p_1 *nodes have no more "children" than* p_2 *nodes*, are expressible by this approach. Seidl et al. [SSMH04] introduce a fixpoint Presburger logic, which, in addition to numerical constraints on children nodes, also supports recursive forward navigation. For example, expressions like *the descendants of* p_1 *nodes have no more "children" than the number of children of descendants of* p_2 *nodes* are allowed. This means that constraints can be imposed on sibling nodes (even if they are deep in the tree) by forward recursive navigation but not on distant nodes which are not siblings.

Compared to the work presented here, neither of the two previous approaches can support constraints like *there are more than* 5 *ancestors of "p" nodes*.

Furthermore, due to the lack of backward navigation, the works found in [DZLM04, SSMH04, DL06] are not suited for succinctly capturing XPath expressions. Indeed, it is well-known that expressions with backward modalities are exponentially more succinct than their forward-only counterparts [OMFB02, GR05].

There is poor hope to push the decidability envelope much further for counting constraints. Indeed, it is known from [KR03, DL06, tCM09] that the equivalence problem is undecidable for XPath expressions with counting operators of the form:

- $\text{PathExpr}_1[\text{count}(\text{PathExpr}_2) = \text{count}(\text{PathExpr}_3)]$, or
- $\text{PathExpr}_1[\text{position}() = \text{count}(\text{PathExpr}_2)]$.

This is the reason why logical frameworks that allow comparisons between counting operators limit counting by restricting the PathExpr to immediate children nodes [DZLM04, SSMH04]. In this paper, we chose a different tradeoff: comparisons are restricted to constants but at the same time comparisons along more general paths are permitted.

6 Conclusion

We introduced a modal logic of trees equipped with (1) converse modalities, which allow to succinctly express forward and backward navigation, (2) a least fixpoint operator for recursion, and (3) cardinality constraint operators for expressing numerical occurrence constraints on tree nodes satisfying some regular properties. A sound and complete algorithm is presented for testing satisfiability of logical formulas. This result is surprising since the corresponding logic for graphs is undecidable [BLMV06].

The decision procedure for the logic is exponential time w.r.t. to the formula size. The logic captures regular tree languages with cardinality restrictions, as

well as the navigational fragment of XPath equipped with counting features. Similarly to backward modalities, numerical constraints do not extend the logical expressivity beyond regular tree languages. Nevertheless they enhance the succinctness of the formalism as they provide useful shorthands for otherwise exponentially large formulas.

This makes it possible to extend static analysis to a larger set of XPath and XML schema features in a more efficient way. We believe the field of application of this logic may go beyond the XML setting. For example, in verification of linked data structures [ZKR08, HIV06] reasoning on tree structures with in-depth cardinality constraints seems a major issue. Our result may help building solvers that are attractive alternatives to those based on non-elementary logics such as SkS [TW68], like, e.g., Mona [KM01].

References

[BLMV06] Piero Bonatti, Carsten Lutz, Aniello Murano, and Moshe Vardi. The complexity of enriched μ-calculi. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP*, volume 4052 of *Lecture Notes in Computer Science*, pages 540–551. Springer-Verlag, 2006.

[CD99] J. Clark and S. DeRose. XML path language (XPath) version 1.0. W3C recommendation, November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[CGS09] Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. Efficient asymmetric inclusion between regular expression types. In *ICDT '09: Proceedings of the 12th International Conference on Database Theory*, pages 174–182, New York, NY, USA, 2009. ACM.

[DL06] S. Demri and D. Lugiez. Presburger modal logic is PSPACE-Complete. In U. Furbach and N. Shankar, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 541–556. Springer, 2006.

[DZLM04] Silvano Dal-Zilio, Denis Lugiez, and Charles Meyssonnier. A logic you can count on. In Neil D. Jones and Xavier Leroy, editors, *POPL*, pages 135–146. ACM, 2004.

[Gel08] Wouter Gelade. Succinctness of regular expressions with interleaving, intersection and counting. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *MFCS*, volume 5162 of *Lecture Notes in Computer Science*, pages 363–374. Springer, 2008.

[GGM09] Wouter Gelade, Marc Gyssens, and Wim Martens. Regular expressions with counting: Weak versus strong determinism. In Rastislav Královic and Damian Niwinski, editors, *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 369–381. Springer, 2009.

[GLS07] P. Genevès, N. Layaïda, and A. Schmitt. Efficient static analysis of XML paths and types. In *PLDI*, pages 342–351, New York, NY, USA, 2007. ACM Press.

[GMN08] Wouter Gelade, Wim Martens, and Frank Neven. Optimizing schema languages for xml: Numerical constraints and interleaving. *SIAM J. Comput.*, 38(5):2021–2043, 2008.

[GR05] Pierre Genevès and Kristoffer Høgsbro Rose. Compiling XPath for streaming access policy. In Anthony Wiley and Peter R. King, editors, *DocEng*, pages 52–54. ACM, 2005.

[HIV06] Peter Habermehl, Radu Iosif, and Tomás Vojnar. Automata-based verification of programs with tree updates. In Holger Hermanns and Jens Palsberg, editors, *TACAS*, volume 3920 of *Lecture Notes in Computer Science*. Springer, 2006.

[HJJ+95] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*, 1995.

[HVP05] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. *ACM Trans. Program. Lang. Syst.*, 27(1):46–90, 2005.

[KM01] Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual*. BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus, January 2001.

[Koz82] D. Kozen. Results on the propositional μ-Calculus. In M. Nielsen and E. M. Schmidt, editors, *ICALP*, volume 140 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 1982.

[KR03] F. Klaedtke and H. Rueß. Monadic second-order logics with cardinalities. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 681–696. Springer, 2003.

[KSV02] O. Kupferman, U. Sattler, and M. Y. Vardi. The complexity of the graded μ-calculus. In A. Voronkov, editor, *CADE*, volume 2392 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 2002.

[KT07] Pekka Kilpelinen and Rauno Tuhkanen. One-unambiguity of regular expressions with numeric occurrence indicators. *Information and Computation*, 205(6):890 – 916, 2007.

[Mar05] Maarten Marx. Conditional xpath. *ACM Trans. Database Syst.*, 30(4):929–959, 2005.

[MLMK05] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704, 2005.

[MS72] Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *FOCS*, pages 125–129. IEEE, 1972.

[OMFB02] Dan Olteanu, Holger Meuss, Tim Furche, and Francois Bry. XPath: Looking forward. In *EDBT '02: Proceedings of the Worshop on XML-Based Data Management*, volume 2490 of *LNCS*, pages 109–127. Springer-Verlag, 2002.

[SSMH04] H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. Counting in trees for free. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer, 2004.

[SV01] Ulrike Sattler and Moshe Y. Vardi. The hybrid μ-calculus. In *IJCAR*, pages 76–91, 2001.

[tCM09] Balder ten Cate and Maarten Marx. Axiomatizing the logical core of XPath 2.0. *Theor. Comp. Sys.*, 44(4):561–589, 2009.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[Var98] M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, pages 628–641, London, UK, 1998. Springer-Verlag.

[ZKR08] Karen Zee, Viktor Kuncak, and Martin Rinard. Full functional verification of linked data structures. In *PLDI*, pages 349–361, New York, NY, USA, 2008. ACM.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399