Automated reasoning on trees with cardinality constraints

Ismael Barcenas Patino

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité: **Informatique**

Arrêté ministériel: 7 août 2006

Présentée par

Ismael Everardo BARCENAS PATIÑO

Thèse dirigée par **Vincent QUINT** et
codirigée par **Nabil LAYAÏDA**

préparée au sein du **Institut National de Recherche en Informatique et en Automatique**
dans **l'École Doctorale Matématiques, Sciences et Technologies de l'Information, Informatique**

Raisonnement automatisé sur les arbres avec des contraintes de cardinalité

Thèse soutenue publiquement le **14 février 2011**,
devant le jury composé de:

Mme. Marie-Christine ROUSSET
Professeur Université de Grenoble, Président
Mme. Véronique BENZAKEN
Professeur Université de Paris Sud 11, Rapporteur
M. Denis LUGIEZ
Professeur Université de Provence, Rapporteur
M. Nabil LAYAÏDA
Chargé de recherche INRIA, Membre

Abstract

Arithmetical constraints are widely used in formal languages like regular expressions, tree grammars and paths. In XML they are used to impose bounds on the number of occurrences described by content models of schema languages (XML Schema, RelaxNG). In query languages (XPath, XQuery), they allow selecting nodes that have a bounded number of nodes reachable by a given path expression. Counting types and paths are thus natural extensions of their countless counterparts already regarded as the core constructs in XML languages and type systems.

One of the biggest challenges in XML is to develop automated procedures for ensuring static-type safety and optimization techniques. To this end, there is a need to solve some basic reasoning tasks that involve constructions such as counting XML schemas and XPath expressions. Every compiler of XML programs will have to routinely solve problems such as type and path type-checking, for ensuring at compile time that invalid documents can never arise as the output of XML processing code.

This thesis studies efficient reasoning frameworks able to express counting constraints on tree structures. It was recently shown that the μ-calculus, when extended with counting constraints on immediate successor nodes is undecidable over graphs. Here we show that, when interpreted over finite trees, the logic with counting constraints is decidable in single exponential time. Furthermore, this logic allows more general counting operators. For example, the logic can pose numerical constraints on number of ancestors or descendants. We also present linear translations of counting XPath expressions and XML schemas into the logic.

Résumé

Les contraintes arithmétiques sont largement utilisées dans les langages formels comme les expressions, les grammaires d'arbres et les chemins réguliers. Ces contraintes sont utilisées dans les modèles de contenu des types (XML Schemas) pour imposer des bornes sur le nombre d'occurrences de nœuds. Dans les langages de requêtes (XPath, XQuery), ces contraintes permettent de sélectionner les nœuds ayant un nombre limité de nœuds accessibles par une expression de chemin donnée. Les types et chemins étendus avec les contraintes de comptage constituent le prolongement naturel de leurs homologues sans comptage déjà considérés comme des constructions fondamentales dans les langages de programmation et les systèmes de type pour XML.

Un des défis majeurs en programmation XML consiste à développer des techniques automatisées permettant d'assurer statiquement un typage correct et des optimisations de programmes manipulant les données XML. À cette fin, il est nécessaire de résoudre certaines tâches de raisonnement qui impliquent des constructions telles que les types et les expressions XPath avec des contraintes de comptage. Dans un futur proche, les compilateurs de programmes XML devront résoudre des problèmes de base tels que le sous-typage afin de s'assurer au moment de la compilation qu'un programme ne pourra jamais générer de documents non valides à l'exécution.

Cette thèse étudie les logiques capables d'exprimer des contraintes de comptage sur les structures d'arbres. Il a été montré récemment que le μ-calcul sur les graphes, lorsqu'il est étendu à des contraintes de comptage portant exclusivement sur les nœuds successeurs immédiats est indécidable. Dans cette thèse, nous montrons que, sur les arbres finis, la logique avec contraintes de comptage est décidable en temps exponentiel. En outre, cette logique fournit des opérateurs de comptage selon des chemins plus généraux. En effet, la logique peut exprimer des contraintes numériques sur le nombre de nœuds descendants ou même ascendants. Nous présentons également des traductions linéaires d'expressions XPath et de types XML comportant des contraintes de comptage dans la logique.

Para Lia Danae

Acknowledgements

Many people contributed to the development of this thesis. I would like to express my gratitude to all of them.

I would like to thank my supervisors Vincent Quint and Nabil Layaïda. They first gave me the opportunity to join the WAM team at INRIA as their student. During the development of my thesis, they also provided me with excellent research conditions.

In the WAM team, I was fortunate to meet Pierre Genevès. From the beginning to the end, this thesis benefited of Pierre's guidance. I would like to thank Pierre for his invaluable availability and support.

I have been honoured to have Véronique Benzaken and Denis Lugiez as referees for this dissertation. I thank them for their insightful comments and suggestions, which helped to improve this dissertation document. I am also grateful to Nils Gesbert for his suggestions.

I would like to also thank Marie-Christine Rousset for taking the role of examiner and president in the thesis committee.

The results described in this dissertation were benefited of a collaboration with Alan Schmitt. I am very grateful to Alan for the great moments doing research.

To the members of the WAM and EXMO teams, in particular to Melisachew Chekol, Freddy Limpens, Chan Le Duc, Jérôme David, Cássia Trojahn dos Santos, Giussepe Pirrò and Manuel Atencia, thank you for providing me a very pleasant and friendly atmosphere at INRIA.

To all my family, Luz María, Everardo, Iliana, and Blanca, thank you for your continuous support.

All my gratitude is for Lia Danae and Cynthia. Thank you for all your sacrifice and support. I would never be able do it without you.

Contents

List of Figures

Chapter 1

Introduction

The topic of this work is the study of formal languages in the context of mathematical logic.

[Kleene, 1956] first defined computational models for formal languages by means of tree automata. Then, it was found in the seminal works of [Büchi, 1960], [Trakhtenbrot, 1957] and [Elgot, 1960], that monadic second order logic is exactly as expressive as tree automata. Since then, logical frameworks have been of great importance in formal language research. The study of more general structures than strings (regular expressions), such as trees or partial orders, is an example where logical frameworks have served to deepen research in formal languages.

[Pnueli, 1977] introduced the use of temporal logic in the verification of programs, and later he also applied logic-based techniques to verify reactive systems [Pnueli, 1985].

[Clarke and Emerson, 1981] and [Queille and Sifakis, 1982] introduced logic as a framework for the model checking problem: given a model system, test automatically if the model satisfies a given specification. Since then, a lot of research has been explored for the model checking of software and hardware systems.

1.1 Motivations

This work is devoted to the study of the satisfiability problem for tree logics: given a logical formula, test automatically whether the formula has a tree model or not. More precisely, we are interested in the development of satisfiability algorithms for logics able to efficiently express counting constraints. We use the algorithm to model several decision problems in formal languages, such as the equivalence and containment of expressions.

Figure 1.1 Syntax of Regular Expressions over a signature Σ

e	:=	ϵ	empty string
	|	p	single symbol of the signature Σ
	|	$e_1 e_2$	concatenation
	|	$e_1 \vert e_2$	alternation
	|	$e^\star$	Kleene star

1.1.1 Counting with regular expressions

Regular expressions were introduced by [Kleene, 1956] and their syntax is in Figure 1.1. A regular expression is interpreted as a set of finite sequences of symbols (strings or words) from a finite alphabet. Consider the following example of a regular expression over the signature (alphabet) $\Sigma = \{p, q\}$:

$$(pq)^\star$$

This expression is interpreted as the following set:

$$\{\epsilon, pq, pqpq, pqpqpq, \ldots, pqpqpqpqpqpq \ldots\}$$

It is well known that some formal languages easily described in English may require voluminous regular expressions. For instance, as pointed out in [Henriksen et al., 1995], the language L_{2a2b} of all strings over $\Sigma = \{a, b, c\}$ containing at least two occurrences of a *and* at least two occurrences of b requires a large expression, such as the following:

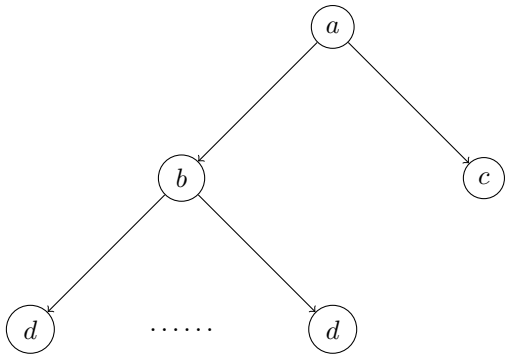
$$\begin{aligned}
&(a|b|c)^\star a(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star \mid \\
&(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star \mid \\
&(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star \mid \\
&(a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star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

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

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

In logical terms, conjunction offers a dramatic reduction in expression size, which is crucial when the complexity of the decision procedure depends on the formula size. More precisely, [Gruber and Holzer, 2009] showed that simple regular languages (without intersection) produce double-exponential larger expressions than regular formalisms equipped with intersection.

Figure 1.2 Instance of the regular tree type $a[b[(d[\epsilon])^\star]c[\epsilon]]$



If we now consider a formalism equipped with the ability to describe numerical constraints on the frequency of occurrences, we get another exponential reduction in size ([Kilpeläinen and Tuhkanen, 2003]). For instance, the above expression can be formulated as follows:

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

In the works of [Kilpeläinen and Tuhkanen, 2007], [Colazzo et al., 2009] and [Gelade et al., 2009b], different extensions of regular expressions with intersection, counting constraints, and interleaving have been considered over strings, and for describing content models of sibling nodes in XML schema languages. The complexity of the inclusion problem over these different language extensions and their combinations typically ranges from polynomial time to exponential space (see [Gelade et al., 2009b] for a survey).

Most XML schema languages, such as DTDs, XML Schema or RELAX NG, can be effectively embedded into regular tree types [Murata et al., 2005].

1.1.2 Counting with regular tree types

Regular tree types can been seen as regular expressions over tree models. See Figure 1.2 for an instance of the following regular tree type expression.

$$a[b[(d[\epsilon])^\star]c[\epsilon]]$$

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, such as a subtree. Suppose, for instance, that we want to define a tree language over Σ where there is no more than 2 "b" nodes. This requires a quite large

regular tree type expression, such as the following:

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

In the expression above (written as a grammar), x_{root} is the starting non-terminal; $x_{\neg b}, x_{b\leq 1}, x_{b\leq 2}$ are non-terminals; the 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. 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.

The problem with counting constraints on regular languages 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, which are intuitive and compact with respect to size. 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 regular languages, it has a drastic impact on succinctness, thus making reasoning over these languages harder. Indeed, reasoning on this kind of counting extensions without relying on their expansion (in order to avoid syntactic blow-ups) is often tricky, as noted by [Gelade et al., 2009a]. Determining satisfiability, containment, and equivalence over these classes of extended regular languages typically requires involved algorithms with higher complexity than ordinary regular languages. Actually, according to [Meyer and Stockmeyer, 1972], the equivalence of regular expressions with squaring (certain kind of counting constraints) requires exponential space to be computed.

1.1.3 Counting with regular paths

The XPath language is the standard query language for XML documents. XPath expressions look like *directory navigation paths* over unraked tree structures. See for instance in Figure 1.4 the tree shaped format of the XML document of Figure 1.3. XPath is also an important part of other XML technologies such as XSLT and XQuery.

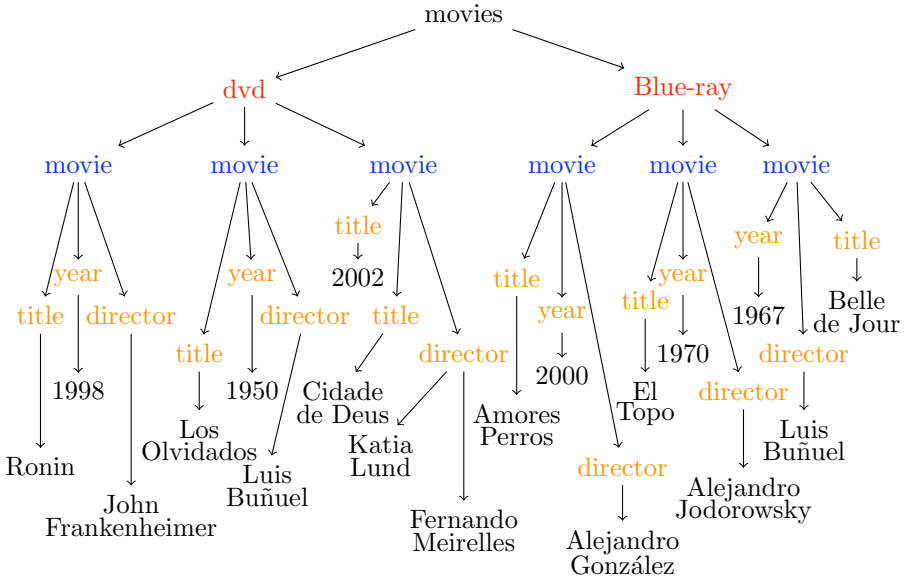
From a given context node, XPath expressions selects subsets of tree nodes. Take for instance the following expression:

/movies/dvd/movie/title[.="Ronin"]

Figure 1.3 Example of a XML document

```
<movies>
     <dvd>
          <movie>
               <title>Ronin</title>
               <year>1998 </year>
               <director>John Frankenheimer </director>
          </movie>
          <movie>
               <title>Los olvidados</title>
               <year>1950 </year>
               <director>Luis Buñuel </director>
          </movie>
          <movie>
               <title>Cidade de Deus</title>
               <year>2002</year>
               <director>Fernando Meirelles</director>
               <director>Katia Lund</director>
          </movie>
     </dvd>
     <blue-ray>
          <movie>
               <title>Amores Perros</title>
               <year>2000</year>
               <director>Alejandro González</director>
          </movie>
          <movie>
               <title>El Topo</title>
               <year>1970</year>
               <director>Alejandro Jodorowsky</director>
          </movie>
          <movie>
               <title>Belle de Jour</title>
               <year>1967</year>
               <director>Luis Buñuel</director>
          </movie>
     </blue-ray>
</movies>
```

Figure 1.4 Tree structure of the XML document of Figure 1.3



The expression navigates from the root *movies* to the child labelled by *dvd*, then it goes to the *movie* children, and it proceeds navigating to the children named *title* that have at least a child labelled by *Ronin*.

One of the reasons why XPath is popular for web programming resides in its ability to express multidirectional navigation. Indeed, XPath expressions may use recursive navigation, to access descendant nodes, and also backward navigation, to reach previous siblings or ancestor nodes. For instance in the previous example, we can select the node named *Ronin* without even knowing the structure of the document in a more compact manner as follows:

/descendant::*[.=”Ronin”]

XPath expressions can also express cardinalities constraints. In contrast with regular expressions and types, cardinality constraints in XPath expressions can be imposed in other nodes than the children ones. For instance, in order to select the titles of the movies with more than 1 directors from the database in Figure 1.4 we can write the following:

/descendant::movie[count(director/*)>1]/title

Note that the counted nodes are not the ones labelled by *director*, but their children nodes, that is, the counted nodes are descendants of the *movie* nodes.

From the works of [Genevès and Rose, 2005], [Demri and Lugiez, 2006] and [ten Cate and Marx, 2009], we know that expressing cardinality restrictions on

nodes accessible by recursive multidirectional paths may introduce an extra-exponential cost, or may even lead to undecidable formalisms.

1.1.4 Research challenges and methodology

Reasoning on expressive logics with counting constraints has been shown to be very hard. Computational complexity ranges from PSPACE to undecidability (see for instance [Demri and Lugiez, 2006]).

Automata machinery has been the most popular computational framework for expressive logics. However, efficient implementations of automata have been very difficult to achieve. For instance, ofter decidability of Monadic Second Order Logic was shown by means of tree automata by [Büchi, 1960], it took around 35 years to implement an efficient automata-based solver [Henriksen et al., 1995].

From works of [Tanabe et al., 2005] and [Genevès et al., 2007], we have learned that certain automata-free techniques for decision procedures of expressive logics can result in efficient implementations.

We aim in this work to develop expressive logics, able to express regular properties as well as counting constraints. We are also particularly interested in the development of computational frameworks for the logics that can be efficiently implemented. For that purpose, we chose to develop automata-free decision algorithms for the logics in the manner of [Genevès et al., 2007].

1.2 Thesis overview

To overcome the challenges involved in the problem of logic-based reasoning with counting constraints, we follow two directions: first we design a tree logic able to efficiently express counting constraints on children nodes in Chapter 3; then, in Chapter 4, we present a logic able to express counting constraints on nodes which are reachable by means of multidirectional recursive paths.

Proper justification for our work is provided in Chapter 2, where a detailed state of the art of works on reasoning in trees with counting constraints is presented.

Conclusion and further research directions are detailed in Chapter 5.

In the remainder of this section we present an overview of the contributions.

1.2.1 Counting on children

We present a tableau-based satisfiability algorithm for the fully enriched alternate-free μ-calculus for trees in Chapter 3. The tree logic is able to express multi-directional recursive navigations and to express numerical constraints on the number of children nodes. Fully enriched μ-calculus was previously shown to be undecidable when interpreted over graph models by [Bonatti and Peron, 2004].

We also show in Chapter 3 how to efficiently encode regular paths and types, with counting constraints on children, into the logic. This results in a reasoning framework for XPath and XML schema languages with counting constraints with an exponential computational cost. No previous work is known

to solve the equivalence of XML schemas with counting constraints on children in exponential time.

1.2.2 Deep counting

In Chapter 4 we extended μ-calculus with counting constraints on regular paths. This logic, in contrast with the one introduced in Chapter 3, can pose counting constraints on further nodes than children ones, as descendant nodes for instance.

Equipping logics with counting constraints on regular paths may produce undecidable formalisms, as noted in the work of [Demri and Lugiez, 2006]. We were able to identify a decidable in exponential time form of deep counting: counting expressions cannot occur in the scope of fixpoints and other counting operators. We also showed in Chapter 4, that the logic is able to capture XPath and XML type expressions with counting constraints on multidirectional recursive paths.

Recall although the logic in Chapter 3 can pose counting constraints only on children, in contrast with the logic of Chapter 4, it is possible for counting formulas to occur in the scope of fixpoints and other counting operators. This makes the two logics incomparable in terms of succinctness.

Chapter 2

Automated Reasoning on Trees

Given an artificial language, by automated reasoning, we mean automatic methods to test the truth status of language expressions. For instance, in the context of formal languages, the equivalence of two regular expressions is a reasoning problem.

Mathematical logic has been extensively used as a formalization tool, and its application as a reasoning framework can be traced back to Leibniz. In this chapter, we present a state of the art of the modern use of mathematical logic in automated reasoning on tree structures.

This chapter is divided in two parts: classical and modal logics. Classical logics represent the older and now more or less well understood setting, and they are often used as context for the study of theoretical issues. Regarding modal logics, there is nowadays an increasing interest in them, mainly because such logics have turned out to have very appealing computational properties.

The study of reasoning frameworks for counting (arithmetical) expressions has been always to much interest of the scientific community due to their importance in many fields. Mathematicians for instance were the first in developing logical frameworks for the analysis of arithmetic. We focus, in this chapter, on studies of reasoning frameworks for tree structures able to express counting constraints. More precisely, we closely follow the application of reasoning frameworks in the analysis of XML programming languages. The static analysis of XPath and XML schema languages is of special interest to us.

2.1 Classical First Order Theories

When David Hilbert raised the question about the consistency of arithmetic, Kurt Gödel used First Order Logic (FOL) as a formal framework in the analysis of arithmetical expressions. It turned out that there is no way to prove the

consistency of arithmetic using the theory of arithmetic itself [Gödel, 1931].

It was later shown, independently by [Church, 1936b] and [Rosser, 1936], that the theory of Peano arithmetic (PA) is undecidable: there is no algorithm able to decide whether an arithmetic expressions is valid or not. On the other hand, the undecidablity of FOL was proven by [Church, 1936a] and [Turing, 1936].

Fortunately, these negative results do not preclude the existence of decidable, and indeed very useful, fragments of both PA and FOL.

2.1.1 Arithmetic

The encoding of Peano postulates (Figure 2.1) about Arithmetic (PA) into FOL resulted in the first order theory of multiplication and addition. As already mentioned, this theory turned out to be undecidable.

[Presburger, 1929] showed that the first order theory of addition, that is, PA without multiplication, is decidable. This theory was called Presburger Arithmetic (PresA). Since then, many other related decidable theories were identified. For instance, [Beltyukov, 1976] and [Lipshitz, 1976] showed that the existential theory of addition and divisibility is also decidable. Among the most recent works, [Bozga and Iosif, 2005] identified another decidable fragment of the theory of addition and divisibility. A detailed survey about decidability results in Arithmetic is provided by [Bés, 2002].

PresA found application in practically all areas of Computer Science. For instance, in Program Verification, balanced tree structures, in contrast with their unbalanced counterpart, require frameworks able to capture counting constraints [Manna et al., 2007]. PresA has been also successfully applied in Program Verification by [Zee et al., 2008] and [Nguyen et al., 2007]. Counting expressions came along with Programming Languages, and PresA has been always represented a strong formal framework candidate.

In the XML setting, we found the works of [Dal-Zilio et al., 2004] and [Seidl et al., 2004]. In these works, logics are extended with Presbuger constraints over tree structures in order to provide an analysis framework for XML.

Regarding the computational complexity, it turned out that PresA is not particularly easy to compute when encoded into FOL. An exponential lower bound was provided by [Fischer and Rabin, 1974], whereas a decision algorithm with a triple exponential upper bound was introduced by [Oppen, 1978].

2.1.2 First Order Logic

First Order Logic (FOL) is a formalism widely used by mathematicians interested in foundations because it is expressive enough to capture two important mathematical theories: Zermelo-Fraenkel Set Theory and Peano Arithmetic [Mendelson, 1964]. Although full FOL was long ago proven to be undecidable by [Gödel, 1931], many decidable fragments turned out to be very useful in many areas of Computer Science, such as the specification and ver-

Figure 2.1 Peano postulates encoded into First Order Logic

$\forall x, y, z \in \mathbf{N} : (x + y) + z = x + (y + z)$	$\forall x, y \in \mathbf{N} : x + y = y + x$
$\forall x, y, z \in \mathbf{N} : (x * y) * z = x * (y * z)$	$\forall x, y \in \mathbf{N} : x * y = y * x$
$\forall x, y, z \in \mathbf{N} : x * (y + z) = (x * y) + (x * z)$	$\forall x \in \mathbf{N} : x + 0 = x * 1 = x$
$\forall x \in \mathbf{N} : x * 0 = 0$	$\forall x \in \mathbf{N} : x + 0 = x$
$\forall x, y, z \in \mathbf{N} : x < y \wedge y < z \Rightarrow x < z$	$\forall x \in \mathbf{N} : \neg(x < x)$
$\forall x, y \in \mathbf{N} : x < y \vee x = y \vee y > x$	$\forall y, y, z \in \mathbf{N} : x < y \Rightarrow x + z < y + z$
$\forall x, y, z \in \mathbf{N} : 0 < z \wedge x < y \Rightarrow x * z < y * z$	$\forall x, y \in \mathbf{N}, \exists z \in \mathbf{N} : x < y \Rightarrow x + z = y$
$0 \leq 1 \wedge \forall x \in \mathbf{N} : x > 0 \Rightarrow 1 \leq x$	$\forall x \in \mathbf{N} : 0 \leq x$

ification of systems and programming languages, artificial intelligence, etc. [Grädel and Otto, 1999].

[Mortimer, 1975] showed the decidability of the fragment of FOL where only binary predicates are allowed, that is, where predicates have only two variables (FOL^2). Much later, it was shown by [Grädel et al., 1997a] that the decidability problem for FOL^2 can be solved in non-deterministic exponential time. Two variable logics and their application in the modelling of transition systems are nicely surveyed by [Grädel and Otto, 1999].

One of the pioneering works in the characterization of XPath by means of FOL is the one of [Benedikt et al., 2003]. However, the XPath fragment considered in this work does not consider the negation operator and sibling axes.

[Gottlob et al., 2005] identified the navigation core of XPath, and what is commonly known as Core XPath. This XPath fragment comprises the main navigational features of XPath: recursive and multi-directional axes, path composition, and the boolean combination of qualifiers (including negation).

[Marx and de Rijke, 2005] provided a FOL characterization of Core XPath. Furthermore, in this last work, an extension of Core XPath was defined in order to be complete with respect to FOL.

A recent work by [Kieronski and Otto, 2005] studies FO^2 enriched with the presence of binary equivalence relations. More precisely, the logic resulting from the addition of exactly one equivalence relation to FO^2 enjoys the finite model property and it is decidable in exponential time. It was also found that the logic extended with two equivalence relations is also decidable. Moreover, when more than 2 equivalence relations are considered, then the logic becomes undecidable.

The consideration of an equivalence relation, in the XML context, can serve as a model for data equality tests, that is, such a relation holds when the content of the two nodes is the same. [Bojanczyk et al., 2009] provided a FO^2 characterization of a XPath fragment with data equality tests. Complexity

results for decidability range from NEXPTIME to 3NEXPTIME.

2.1.3 Counting in First Order Logic

[Grädel et al., 1997b] showed the decidability of the logic C^2 resulted from extending FO^2 with numerical constraints.

For instance, in C^2 we can write the following formula, where k is a natural number.

$$\forall x \exists^{\leq k} y \phi(x, y)$$

If ϕ is thought of as the edge relation of graph structures, the formula denotes all graphs whose degree is bounded by k.

C^2 was shown not to have the finite model property, that is, some formulas may only be satisfiable over infinite models. The problem of deciding whether a formula has a finite model or not is known as finite satisfiability.

Regarding the complexity of C^2, [Pacholski et al., 1997] showed that both satisfiability and finite satisfiability have a non-deterministic exponential cost. However, the assumption of a unary encoding of numbers in formulas was assumed. Note that a binary representation of numbers provides exponentially more succinct formulas. Hence, from the work of [Pacholski et al., 1997], we can only obtain a non-deterministic doubly exponential complexity bound for C^2.

It was only until the work of [Pratt-Hartmann, 2005], where a non-deterministic single exponential bound was provided for C^2. In this later work, a binary encoding of numbers in formulas was considered.

The main motivation for study of second order theories to model formal languages came with the classical result of [Büchi, 1960], that established the exact correspondence between regular languages (those definable by tree automata) and Monadic Second Order Logic (MSO).

2.2 Classical Second Order Theories

The notion of regular expressions, widely known in Computer Science, was originally introduced by [Kleene, 1956]. More precisely, Kleene characterized finite regular expressions (words or strings) in terms of finite automata.

In a different research direction, [Church, 1957] explored the connection between automata theory and arithmetic. Church introduced a method that solves the problem of circuit synthesis by means of some restricted systems of arithmetic. Such arithmetical systems turned to be expressible in terms of Monadic Second Order Logic (MSO). In order to provide a computational model for the arithmetical sytems, Church aimed an automata framework for MSO.

The automata machinery for MSO was developed by [Trakhtenbrot, 1957], [Büchi, 1960], and [Elgot, 1960]. Consequently, due to its relationship with tree automata, finite regular expressions were also characterized by MSO formulas.

Figure 2.2 Second Order Induction Axiom

$$\forall X : ((0 \in X \wedge \forall x : (x \in X \Rightarrow (x+1) \in X)) \Rightarrow \forall x : (x \in X))$$

2.2.1 Monadic Second Order Logic

In Second Order Logic, besides first order variables ranging over nodes, there are second order variables ranging over relations. See for instance the second order formula of Figure 2.2, which is part of the encoding of Peano Arithmetic into Second Order Logic.

In Monadic Second Order Logic (MSO), second order variables can only be quantified over unary relations, that is, sets. When second order quantification is restricted to finite sets, then we call the logic Weak Monadic Second Order Logic.

[Trakhtenbrot, 1957], [Büchi, 1960], and [Elgot, 1960] showed the decidability of Weak Monadic Second Order Logic with only one succesor relation (WS1S), that is, where models are finite strings. Later, [Büchi, 1962] showed decidability for S1S: when infinite strings models are considered.

Weak Monadic Second Order Logic with two succesor relations, that is, where models are finite binary trees, is called WS2S. The decidability of WS2S was shown by [Thatcher and Wright, 1968], and later [Doner, 1970]. [Rabin, 1969] proved decidability of S2S: the case with infinite tree models.

Using symbolic techniques [Bryant, 1986] and tree automata, the Mona solver for WS1S and WS2S was implemented by [Henriksen et al., 1995] and [Elgaard et al., 1998]. These results were applied to the verfication of linked data structures by [Jensen et al., 1997] and [Møller and Schwartzbach, 2001]. Also [Biehl et al., 1996] reported an application of Mona to arithmetic.

A translation of Core XPath (including backward navigation) into WS2S was first presented in [Genevès and Layaïda, 2007]. Also experimental results on XPath decision problems with the MONA solver were reported in this last work.

2.2.2 Counting in Second Order Logic

[Courcelle, 1990] introduced an extension of Monadic Second Order Logic with counting quantifiers, called CMSOL. Since quantification ranges over both nodes and sets of nodes, then also the cardinality constraints are applied over nodes and sets of nodes. For this work only finite models were considered, and it was also shown that MSO is strictly contained in CMSOL.

Another kind of counting in MSO was introduced by [Bárány et al., 2009]. Finitely branching tree models with a countable height are considered by this logic. In this approach, cardinality constraints can be imposed only over sets of nodes. Moreover, this logic can only impose constraints with cardinalities

greater than the cardinality of natural numbers. To illustrate the use of such kind of constraints consider the following expression:

There exist uncountably many subsets of nodes satisfying some property expressed by a MSO formula.

[Seidl et al., 2003] designed a logic combining WS2S with Presburger Arithmetic, where the cardinality constraints are imposed only on children nodes. The following expression can be succinctly encoded by this logic:

Nodes labelled X have less children than nodes labelled Y

The logic in the work of [Seidl et al., 2003] was introduced to support reasoning on XML documents, that is, unranked trees. Unfortunately, the logic turned out to be undecidable.

One of the drawbacks of WS2S is its high complexity for decidability, which was reported to be non-elementary by [Stockmeyer, 1974] and [Meyer, 1975]. Modal logics have been identified as fragments of FOL and MSOL with nice computational properties, which places them as more attractive alternatives.

2.3 Modal logics

Modal logic is the logic resulting from propositional logic enhanced with the modal operators of *necessity* $\Box$ and *possibility* $\Diamond$. The relational semantics of modal logic was independently developed by Hintikka, Kanger and Kripke, and its current form was introduced in [Kripke, 1963]. The intuition behind what is known as Kripke semantics for modal logic is that the truth status of formulas is related with nodes in a graph structure. Then, *necessity* means true at *all* accessible nodes, and *possibility* means true at *some* accessible nodes. See for instance the interpretation of the modal operators, according to a Kripke semantics, in Figure 2.3.

Modal logics have two nice computational properties: great expressive power and low computational cost. Propositional modal logic was shown to have a PSPACE-complete complexity by [Ladner, 1977].

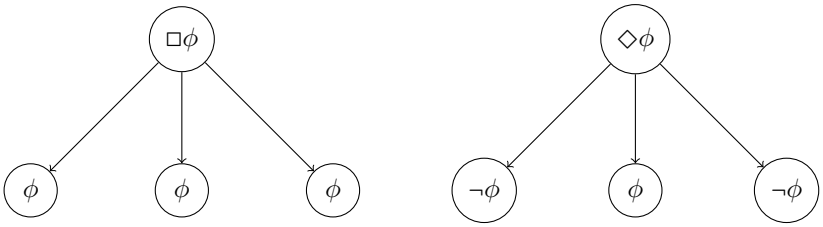
A possible explanation for the nice computational behaviour of modal logic was given by [van Benthem, 1976]. He showed that propositional modal logic represents the fragment of FOL^2 that cannot distinguish bisimilar models.

One of the pioneering works exploiting the nice computational features of modal logics was done by [Pnueli, 1977]. It was shown in that work that the verification of some program properties can be efficiently achieved using temporal logic.

2.3.1 Temporal logics

Temporal logic, or linear temporal logic (LTL), is the modal logic where models are restricted to strings. If models are tree-shaped, then we call such a logic the branching time temporal logic. Several extensions of modal logic with other logical operators have been proposed.

Figure 2.3 Modal formulas



Computational Tree Logic (CTL) is the branching time temporal logic introduced by [Clarke and Emerson, 1981]. CTL includes the temporal operators X,F,G, and U, which mean *next, finally, globally,* and *until*, respectively. These operators can only occur immediately after the path quantifiers A and E, which mean, respectively, *for all* and *there exists* .

For instance, consider the following expression.

There exists a path (sequence of nodes) starting from the current node, such that p holds in all nodes of the path except in the last one, where actually q holds.

Such expression is denoted by the following formula.

$$EpUq$$

Relationships between an existential fragment of CTL and some XPath fragments were studied by [Miklau and Suciu, 2004] and [Gottlob et al., 2005]. XPath fragments decidable in exponential time are developed there. However, the capability to perform bi-directional navigation is not supported by those XPath fragments due to the lack of temporal operators for the past in the logics.

[Emerson and Halpern, 1986] introduced an extension of CTL called CTL*. In this extension, the quantifiers and temporal operators can be freely mixed, in contrast with CTL, where temporal operators can only occur immediately after a path quantifier.

[Laroussinie and Schnoebelen, 2000] and [Reynolds, 2000] developed an extension of CTL* with temporal operators for the past called PCTL*.

Extensive studies on the use of CTL*, PCTL* and other CTL* extensions in the semi-structured data context have been described in the work of [Barceló and Libkin, 2005]. In particular, they showed that all such CTL* extensions are embeddable into FOL^2. Therefore, these formalisms are unable to capture regular tree languages.

2.3.2 Propositional dynamic logic and μ-calculus

In order to add second order properties to temporal logics, [Pratt, 1976] and [Fischer and Ladner, 1979] developed what is called Propositional Dynamic Logic (PDL).

PDL is a multimodal logic where the transitions in models are labelled. This labels are called modalities. Instead of the usual box and diamond operators, operators $[\alpha]$ and $\langle\alpha\rangle$ are considered, where α is a regular expression composed by modalities. Operator $[\alpha]$ and $\langle\alpha\rangle$ denote *all* or *some* paths (sequences of modalities) in α, respectively. Consider for instance the following formula.

$$\langle 1^\star\rangle\phi$$

When interpreted the formula over a model, it denotes the nodes that can access, by means of a path in $1^\star$, a node where ϕ holds. Figure 2.4 depicts a graphical representation of the example.

In [Pratt, 1978] and [Pratt, 1980], PDL decidability is shown to be EXPTIME-complete. A PDL version for finite ordered trees was introduced by [Kracht, 1995]. [Afanasiev et al., 2005] revisited this PDL version and identified some first and second order fragments.

μ-calculus, introduced in its current form in [Kozen, 1983], is a propositional multimodal logic extended with least μ and greatest fixpoints ν. In contrast with PDL, where regular expressions of modalities are allowed, in μ-calculus only single modalities are allowed. Finite and infinite recursive navigation is achieved by means of the fixpoints. These fixpoints provide a great expressive power: actually, μ-calculus turned out to encompass all program-based logics, that is, all families of linear temporal logics, CTL and PDL. As an example, consider the following μ-calculus formula.

$$\mu x.\phi \vee \langle 1\rangle x$$

Since this formula is equivalent to the PDL formula $\langle 1^\star\rangle\phi$, a graphical interpretation is also found in Figure 2.4.

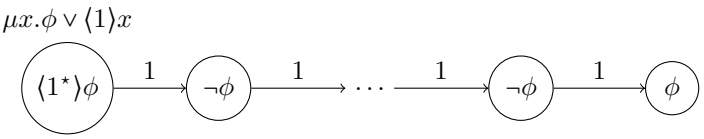
[Arnold and Niwinski, 1992] showed that the expressive power of WS2S corresponds exactly to the expressive power of the μ-calculus fragment where there is no alternation between fixpoints, that is, the alternation-free fragment. In the case of infinite tree models, it was shown by [Janin and Walukiewicz, 1996] that the full μ-calculus has the same expressivity as the MSOL fragment unable to distinguish bisimilar models.

Regarding the complexity of μ-calculus, it is EXPTIME-complete, the lower bound comes from [Fischer and Ladner, 1979], and the upper bound was shown by [Emerson and Jutla, 1988].

The great expressive power and low computational complexity of μ-calculus have made this logic an attractive reasoning framework and many extensions have been studied. [Vardi, 1998] showed that adding converse modalities to μ-calculus yields an exponentially more succinct logic which is still decidable in EXPTIME. These converse modalities allow capturing backward navigation along the tree models.

Due to the close relationship between μ-calculus and MSOL, the most popular computational machinery developed for μ-calculus has been tree automata. Despite the elegance and power of this automata theory, it turned out to be very hard to implement such automata-based decision procedures. It

Figure 2.4 A PDL (μ-calculus) formula



was not before the work of [Tanabe et al., 2005] that a decision procedure for the satisfiability of the alternation-free fragment of μ-calculus with converse was successfully implemented. This tableau-based satisfiability algorithm has an upper bound complexity of $2^{\mathrm{O}(n \log n)}$, where n is the original formula size. [Genevès and Layaïda, 2006] showed that Core XPath and regular tree types can been embedded in this μ-calculus version.

In [Genevès et al., 2007], it was shown that reasoning over trees with the alternation-free fragment of μ-calculus can be more efficient. They developed and implemented a tableau-based satisfiability algorithm with a single exponential time complexity $2^{\mathrm{O}(n)}$. This implementation offers an efficient static analysis system for XPath decision problems in the presence of XML types [Genevès and Layaïda, 2010b].

Automata-based computing machinery for the XPath fragment considered in the work of [Genevès et al., 2007] was later independently reported in the works of [Libkin and Sirangelo, 2010] and [Calvanese et al., 2009].

In [Libkin and Sirangelo, 2010], they introduce a temporal logic for trees with the operators *next* and *until*, and their dual backward navigation operators. Tree automata techniques were developed to compute the satisfiability and model-checking of the logic.

[Calvanese et al., 2009] provided a direct XPath translation into tree automata. In both cases, [Libkin and Sirangelo, 2010] and [Calvanese et al., 2009] the complexity of the satisfiability algorithms is single exponential.

2.3.3 Equality

The extension of modal logics with an operator that can perform equality tests was first introduced by [Alur and Henzinger, 1989]. This operator, called freeze quantification, stores in a register the equivalence class of nodes. In [Alur and Henzinger, 1989], such a freeze quantification operator is considered for linear temporal logic. The navigation involved in the freeze operator considered in this work is limited to the forward direction.

It was shown by [Demri and Lazic, 2006] that freeze quantification in linear temporal logic with finite models is decidable but not primitive recursive, that is, the problem is not decidable in either primitive recursive time or space. The logic becomes undecidable if it is extended with one of the following features: infinite models; past operators (converse modalities); more than one register.

[Lazic, 2006] showed that the safety fragment of linear temporal logic extended with freeze quantification over infinite models is decidable in exponential

memory space.

Freeze quantification has been studied also in the scope of branching time temporal logics. [Jurdzinski and Lazic, 2007] proposed an extension of μ-calculus with freeze quantification. This logic is shown to be decidable, but with a non-primitive recursive complexity. Data equality test in a forward fragment of XPath (with no backward navigation) is also shown to be embeddable into the logic. DTDs are also shown to be captured by the logic. Hence, satisfiability of the forward fragment of Core XPath with data equality test in the presence of DTDs is also decidable.

2.3.4 Graded modalities

Graded modalities are a generalization of the modal operators, where the number of transitions among nodes is bounded by some constant. For instance consider the following formula:

$$\Diamond^{>k}\phi$$

This formula holds on nodes if and only if there are more than k accessible nodes where ϕ holds. Pioneer works introducing the idea of graded modalities are the ones of [Goble, 1970] and [Fine, 1972]. An example of a formula of μ-calculus with graded modalities is depicted in Figure 3.7.

The complexity of the satisfiability problem of propositional modal logic, extended with graded modalities, is known to be PSPACE. This result was shown by [Hollunder and Baader, 1991] when numbers occurring in graded modalities are encoded in unary. The same upper bound holds even when numbers are coded in binary [Tobies, 1999].

[Kupferman et al., 2002] showed that the addition of graded modalities to μ-calculus comes at no additional cost, that is, the satisfiability problem remains in EXPTIME.

[Bonatti et al., 2006] studied the logic obtained when graded μ-calculus is enriched with converse modalities. The cost of satisfiability for this logic, called full graded μ-calculus, is not affected, that is, it is also in EXPTIME.

A richer logic was introduced in [Bonatti and Peron, 2004]. The logic, called fully enriched μ-calculus, has the following features:

- fixpoints for finite and infinite recursive navigation;
- converse modalities for backward navigation;
- graded modalities for numerical constraints over immediate neighbour nodes; and
- nominals, proposition interpreted as singletons, that is, propositions that hold at a single node in the models.

This logic turned out to be undecidable.

In Chapter 3, we show that the alternation-free fragment of this fully enriched μ-calculus is decidable in exponential time when it is interpreted over finite trees. A tableau-based algorithm is proposed for the satisfiability problem.

Furthermore, we show how this logic can be used to solve decision problems within XPath, XML types and regular expressions where numerical restrictions over the occurrence of nodes are present.

Counting constraints over immediate forward nodes, similar to graded modalities, were added to CTL* by [Moller and Rabinovich, 2003]. This graded version of CTL* was found to have the same expressive power than Monadic Path Logic (MPL).

Recently, a more general kind of counting was independently studied by [Ferrante et al., 2009] and [Bianco et al., 2009]. Instead of counting over simple modalities, in their approach, they impose numerical constraints over paths. More precisely, CTL was extended with counting constraints over paths. For example, the following expression can be expressed by this logic:

From the current node, there are more than k forward paths satisfying ϕ_1 in all nodes but the last one, where ϕ_2 holds

Furthermore, [Bianco et al., 2009] showed that this graded version of CTL is exponentially more succinct than the graded μ-calculus. For instance, restricting the number of descendant nodes in a tree requires exponentially larger formulas in the graded μ-calculus than in graded CTL.

Since the complexity of decision procedures for logics is mainly parametrized by the formula size, then an exponential gain in succinctness implies an exponentially more efficient procedure.

In both cases, [Ferrante et al., 2009] and [Bianco et al., 2009], exponential satisfiability algorithms are provided, although an unary encoding of numbers in formulas is assumed.

Also note that the binary encoding of numbers is exponentially more succinct than the unary encoding.

We present in Chapter 4 a μ-calculus for trees with counting constraints over paths. With respect to [Ferrante et al., 2009] and [Bianco et al., 2009], our logic presents the following advantages:

- The fixpoint of μ-calculus can perform more general recursive navigations. μ-calculus encompasses other logics, such as CTL, CTL* and PDL.
- Backward navigation is supported by means of converse modalities. As noted by [Vardi, 1998], converse modalities provide an exponential gain in succinctness compared to only forward modalities.
- The paths constrained by the counting operators are expressed by regular expressions formed by modalities, similar to PDL.
- Since the logic supports converse modalities in the counting constraints, the paths involved in the numerical restrictions can be recursive and multi-directional.
- The numbers occurring in the counting constraints can be encoded in binary.

- A tableau-based satisfiability algorithm with single exponential complexity.

We also provide a linear embedding of XPath and XML type expressions with counting constraints. Hence, the logic can be used as a static analysis framework for XML.

2.3.5 Presburger constraints

The extension of Modal logics with Presburger formulas provides a more general way to impose counting constraints than graded modalities. Consider for instance the following expression.

The current node has more children named p than children named q

Modal logics enriched with Presburger formulas in normal form are introduced by [Dal-Zilio and Lugiez, 2003] and [Dal-Zilio et al., 2004]. This logics, called Sheaves logics, were designed for unraked trees in order to model XML documents with counting constraints. This logics are also extended with regular constraints, in the sense of regular expressions (tree automata).

In order to ensure decidability of Sheaves logics, Presburger constraints can only be applied to children nodes.

In the logic introduced in [Dal-Zilio and Lugiez, 2003], a shuffle or interleaving (&) operator was also supported. As a consequence, a DTD XML type language with the shuffle operator could be characterized via this Sheaves logic.

[Cardelli and Gordon, 2000] designed a modal logic for mobile ambients, for instance, spatially distributed systems. [Dal-Zilio et al., 2004] showed decidability of a fragment of this ambient logic by means of the Sheaves logic supporting Presburger constraints. The complexity for Sheaves logic satisfiability was shown to be non-elementary computational by means of a tree automata.

An extension of propositional modal logic extended with Presburger and regular constraints is reported on [Demri and Lugiez, 2006]. Decidability for the logic was achieved by means of a tableau-based satisfiability-algorithm with a PSPACE complexity.

The complexity of the Sheaves logic in [Dal-Zilio et al., 2004] was improved to PSPACE via a translation into the logic of [Demri and Lugiez, 2006].

PDL for trees, as presented by [Afanasiev et al., 2005], enriched with Presburger constraints was shown to be undecidable also by [Demri and Lugiez, 2006].

[Seidl et al., 2004] developed a modal logic for trees with the following features:

- Presburger constraints on children nodes;
- regular constraints; and
- a fixpoint operator.

In contrast with the work of [Dal-Zilio et al., 2004] and [Demri and Lugiez, 2006], the logic reported by [Seidl et al., 2004] additionally considers a fixpoint operator which enables it to perform finite recursive navigation in the same manner as the least fixpoint of μ-calculus. Consider the following statement in order to illustrate the expressive power of the logic.

The current node has a descendant node with less children labelled by p than children labelled by q

In contrast with the logics proposed here, the works of [Dal-Zilio et al., 2004], [Demri and Lugiez, 2006] and [Seidl et al., 2004], cannot support converse navigation. This lack makes it hard for the logic to be used as a reasoning framework for XPath, where backward navigation is essential for expressing all navigational axes, as well as qualifier predicates.

Note that the logic proposed in Chapter 4 offers at least a double-exponential gain in succinctness with respect to current approaches [Dal-Zilio et al., 2004] , [Demri and Lugiez, 2006] and [Seidl et al., 2004]: multi-directional navigation and counting on recursive paths.

2.4 Conclusion

We have reviewed in this Chapter the state of the art of logical frameworks for automated reasoning on transition systems. The focus was on logics for tree-shaped structures and their application to the static analysis of XML documents. Since the topic of this work is to study the capability of logics of expressing counting constraints, the state of the art on logics with such features was in particular reviewed in this Chapter.

The first lesson we have learned so far is that it is required for the logical frameworks to express certain second order properties (for instance, MSOL) in order to properly capture regular tree languages (XML types).

We have also seen that the use of reasoning frameworks of classical logics serves as an excellent reference context for the study of some fundamental aspects, such as expressivity. However, the practical implementation of reasoning algorithms for classical logics does not seem appealing due to their high complexity ([Meyer, 1975]).

In order to pursue more efficient decision procedures, we chose to study modal logics, which seem more attractive for us as reasoning frameworks due to their great expressive power combined with their relatively low computational analysis cost.

The logical languages developed in this work do not go beyond the expressive power of regular tree languages. On the other hand, the logics are equipped succinct notations for counting constraints and multi-directional recursive navigation over tree models. The gain in succinctness of our languages represents exponentially more efficient reasoning algorithms with respect to current known approaches.

Chapter 3

Fully enriched μ-calculus for trees

The *fully enriched μ-calculus* [Bonatti et al., 2006] is a modal logic with the following features: converse modalities, for backward navigation; least and greatest fixpoints, for finite and infinite recursive navigation; graded modalities, to describe numerical constraints on contiguous nodes; and nominals, to denote singleton node sets. [Bonatti and Peron, 2004] showed that the fully enriched μ-calculus is undecidable. We show in this Chapter that this undecidability result does not extend to finite tree models, that is, we show that the fully enriched μ-calculus for trees is decidable.

[Genevès et al., 2007] developed a tableau-based algorithm for the satisfiability of the μ-calculus with converse and without graded modalities. The complexity of this algorithm was shown to be single exponential. Here we show that adding graded modalities comes at no cost. For that purpose, we develop a tableau-based algorithm for the satisfiability of the fully enriched μ-calculus. We then prove that the satisfiability algorithm is correct and that its complexity is in single exponential time.

The fragment of the XML query language XPath, known as Core XPath, has the following main features:

- Multi-directional and recursive navigation, as expressed by the relations: child, parent, descendant, ancestor.

- Boolean path filters, known as qualifiers, which are boolean predicates witnessing the existence/absence of boolean combinations of paths.

Since the μ-calculus captures recursive and multi-directional navigation on trees, it represents an ideal candidate for an efficient reasoning framework for

Core XPath. A linear embedding of Core XPath into μ-calculus has already been shown by [Genevès et al., 2007].

In the work of [ten Cate and Marx, 2009] it was shown that extending Core XPath with counting expressions, with all the navigational features, can lead to undecidable languages. We introduce in Section 3.5 a first extension of Core XPath with counting constraints on children nodes. We then show that the graded modalities in the fully enriched μ-calculus can efficiently capture such counting constraints. We use this result to introduce a decidable extension of XPath, with counting constraints on children, in EXPTIME. This result leads us to identifying an extension decidable in EXPTIME of Core XPath with counting constraints on children.

In the XML context, structural constraints on documents are expressed by schema languages, such as DTDs, XML Schema, and RELAX NG. A generalization of such schema languages, that is, a language strictly more expressive than them, is defined by Regular Tree Types. These regular types can be seen as the tree-model version of regular expressions over words. We also provide in this Chapter a linear translation of XML types with counting constraints into the fully enriched μ-calculus.

The embedding of XPath and XML type expressions in the fully enriched μ-calculus, together with the satisfiability algorithm, offers an efficient static analysis framework for XPath and XML types with counting constraints on children.

3.1 Trees

In this Section, we give a formal introduction of trees represented as Kripke structures in the style of Modal Logics. For this purpose, before proceeding to the definition of trees, we first define Kripke structures.

3.1.1 Kripke Structures

The goal of this Section is to give a relational semantics to modal logics as introduced by [Kripke, 1963]. Kripke structures considered here are relational structures representing connected graphs with a countable number of nodes.

Definition 3.1.1 (Signature)**.** A *signature* is a tuple $(\mathcal{P}, \mathcal{M})$, such that:

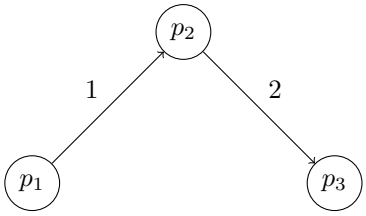
- $\mathcal{P}$ a is countable set of *propositions*; and
- $\mathcal{M}$ is a countable set of *modalities*.

We will name *nominals* some special propositions.

Definition 3.1.2 (Kripke Structure)**.** Given a signature $(\mathcal{P}, \mathcal{M})$, a Kripke structure K is defined as a tuple $(\mathcal{N}^K, \mathcal{R}^K, \mathcal{L}^K)$, such that:

- $\mathcal{N}^K$ is a countable set of *nodes*;

Figure 3.1 A Kripke Structure



- $\mathcal{R}^K$ is a set of binary relations $\mathcal{R}^m : \mathcal{N}^K \times \mathcal{N}^K$ for each $m \in \mathcal{M}$, we write $\mathcal{R}^K(n_1, m) = n_2$ when $(n_1, n_2) \in \mathcal{R}^m$; and
- $\mathcal{L}^K$ is a total and injective function $\mathcal{N}^K \mapsto \mathcal{P}$, in the case of non nominal propositions, that is, a labelling function that marks every node with exactly one proposition. For the case of nominals, the function is not required to be total nor injective.

When the structure K is clear from the context, we often omit the superscript.

Figure 3.1 is a graphical representation of the Kripke structure $K = (\mathcal{N}, \mathcal{R}, \mathcal{L})$ such that:

- the signature $(\mathcal{P}, \mathcal{M})$ is defined by
  - $\mathcal{P} = \{p_1, p_2, p_3\}$;
  - $\mathcal{M} = \{1, 2\}$;
- $\mathcal{N} = \{n_1, n_2, n_3\}$;
- $\mathcal{R}(n_1, 1) = n_2$, and $\mathcal{R}(n_2, 2) = n_3$; and
- $\mathcal{L}(n_i) = p_i$, where $i = 1, 2, 3$.

Definition 3.1.3 (Converse modalities)**.** A Kripke structure K with *converse modalities* is a Kripke structure where:

- the set of modalities is partitioned in two subsets $\mathcal{M}$ and $\overline{\mathcal{M}}$; and
- for all distinct pairs of nodes n_1, n_2, $\mathcal{R}^K(n_1, m) = n_2$, if and only if, $\mathcal{R}^K(n_2, \overline{m}) = n_1$, where $m \in \mathcal{M}$ and $\overline{m} \in \overline{\mathcal{M}}$.

$m \in \mathcal{M}$ can also be written $\overline{\overline{m}}$. Modalities in $\overline{\mathcal{M}}$ are called *converse modalities*.

As an example, consider the Kripke structure of Figure 3.1. Then, the Kripke structure with converse must satisfy:

- $\mathcal{M} = \{1, 2, \overline{1}, \overline{2}\}$; and
- $\mathcal{R}(n_1, 1) = n_2$, $\mathcal{R}(n_2, 2) = n_3$, $\mathcal{R}(n_2, \overline{1}) = n_1$, and $\mathcal{R}(n_3, \overline{2}) = n_2$.

3.1.2 Tree Structures

We first define trees in terms of Kripke structures. Then we show that there is a bijection between binary trees and trees with arbitrary finite degree.

A tree is a connected graph with no cycles. In contrast with ranked trees, where there is an a priori bound on the number of children for each node, we consider here unranked trees, trees with no such bound.

Definition 3.1.4 (Trees)**.** A *tree structure* or simply a *tree* is a Kripke structure K with converse modalities satisfying the following properties:

- for every node n, except one (the root), there is exactly one parent, that is, there is exactly one converse modality $\overline{m}$, such that $\mathcal{R}^K(n, \overline{m}) = n'$ holds;
- for exactly one node n, the *root*, there is no parent, that is, there is no converse modality $\overline{m}$, such that $\mathcal{R}^K(n, \overline{m}) = n'$ holds.

If $\mathcal{R}^K(n_1, m) = n_2$, where m is a non-converse modality, we say that n_1 is the parent of n_2, and that n_2 is the child of n_1. A *binary tree* is a tree where every node has no more than two children.

As already noted by [Hosoya and Pierce, 2003] and [Neven, 2002], there is a straightforward isomorphism between n-ary trees and binary trees. In a binary tree, one transition is interpreted as the first child relation, whereas the second transition is interpreted as the following sibling relation.

Definition 3.1.5 (Bijection among n-ary trees and binary trees)**.** Consider a signature $(\mathcal{P}_1, \mathcal{M}_1)$ of a k-ary tree K_1, that is, where every node has at most k children. We define the equivalent binary tree K_2 of K_1 as follows:

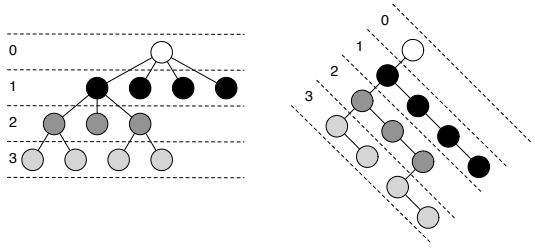
- the signature $(\mathcal{P}_2, \mathcal{M}_2)$ of K_2 is:
  - $\mathcal{P}_2 = \mathcal{P}_1$; and
  - $\mathcal{M}_2 = 1, 2, \overline{1}, \overline{2}$;
- $\mathcal{N}^{K_2} = \mathcal{N}^{K_1}$; $\mathcal{L}^{K_2} = \mathcal{L}^{K_1}$; and
- for every node n, such that $\mathcal{R}^{K_1}(n, m_i) = n_i$ for $i = 1, \ldots, l$ and m_i is not a converse modality, we have that $\mathcal{R}^{K_2}(n, 1) = n_1$ and $\mathcal{R}^{K_2}(n_j, 2) = n_{j+1}$ for $j = 1, \ldots, l-1$.

An equivalent n-ary tree of a binary tree is defined analogously.

Figure 3.2 depicts a graphical representation of the bijection among n-ary trees and binary trees.

Therefore, from now on, without loss of generality, we will only consider binary trees.

Figure 3.2 n-ary to binary trees



3.2 The tree logic

We now introduce a fully enriched μ-calculus for trees. The alternation-free μ-calculus, originally introduced by [Genevès et al., 2007], is extended with graded modalities. The main features of the modal tree logic presented here are the following:

- a fixpoint operator, that allows recursive navigation;
- converse modalities, which allow backward navigation;
- graded modalities, which allow counting constraints on children nodes; and
- nominals, which denote singleton subsets of the tree model.

3.2.1 Syntax

We assume a fixed signature $(\mathcal{P}, \mathcal{M})$, such that $\mathcal{M} = \{1, 2, \overline{1}, \overline{2}\}$. 1 is interpreted as the first child relation, the dual $\overline{1}$ is considered as the parent relation of the first child. 2 is the following sibling relation, whereas the dual $\overline{2}$ is the previous sibling relation.

We also consider a fixed countable set of variables X.

Formulas of the μ-calculus are basically composed by propositions in the base case. Boolean combinations are also considered: negation, conjunction and disjunction of formulas. A modal and a n-ary fixpoint operators are also considered. When we write $\overline{x}$ and $\overline{\phi}$, we mean finite sequences $x_1, \ldots, x_k$ and $\phi_1, \ldots, \phi_k$, respectively. We also consider two types of graded formulas: *the greater than* $\phi^{>k}$, and *the less or equal than* $\phi^{\leq k}$, where k is a natural number. Detailed syntax in negation normal form is given in Figure 3.3.

The negation normal form of a formula is computed by the rules of Figure 3.4, where $\phi\left[{}^{x_1}/_{x_2}\right]$ means that every occurrence of x_2 in ϕ is replaced by x_1, $\phi\left[{}^{\overline{x_1}}/_{\overline{x_2}}\right]$ is the n-ary version of $\phi\left[{}^{x_1}/_{x_2}\right]$, and $\neg\overline{x} = \neg x_1, \ldots, \neg x_k$.

We also consider the syntactic sugar from Figure 3.5.

Figure 3.3 Enriched μ-calculus syntax in negated normal form

$\Phi \ni \phi$	$:=$	p	proposition
		$\top$	true constant
		x	recursion variable
		$\neg p$	negated proposition
		$\neg\top$	false constant
		$\neg\langle m\rangle\top$	no transition
		$\phi \vee \phi$	disjunction
		$\phi \wedge \phi$	conjunction
		$\langle m\rangle\phi$	modal formula
		$\mu\overline{x}.\overline{\phi}$ in ϕ	least n-ary fixpoint
		$\phi^{\#k}$	graded formula

Figure 3.4 Negation Normal Form of Formulas in Enriched μ-calculus

$\neg\langle m\rangle\phi$	$\equiv$	$\neg\langle m\rangle\top \vee \langle m\rangle\neg\phi$
$\neg\mu\overline{x}.\overline{\phi}$ in ϕ	$\equiv$	$\mu\overline{x}.\overline{\neg\phi\left[{}^{x}/_{\neg x}\right]}$ in $\phi\left[{}^{\overline{x}}/_{\neg\overline{x}}\right]$
$\neg\phi^{>k}$	$\equiv$	$\phi^{\leq k}$
$\neg\phi^{\leq k}$	$\equiv$	$\phi^{>k}$

Figure 3.5 Syntactic sugar for enriched μ-calculus

$\neg\phi$	$\equiv$	negated normal form of ϕ
$\mu x.\phi$	$\equiv$	$\mu x.\phi$ in x
$\phi^{=k}$	$\equiv$	$\phi^{\leq k} \wedge \phi^{>k-1}$

3.2.2 Cycle-freeness

We will only consider cycle-free formulas. The notion of *cycle-freeness* was originally introduced by [Genevès et al., 2007]. The purpose of cycle-free formulas is to make least and greatest fixpoints coincide. For that reason we do not include the greatest fixpoint in the syntax of the logic. Intuitively, a cycle-free formula is a formula where the fixpoint variables do not occur under the scopes of both a modality and its converse. For instance, the following formula is not cycle-free: $\mu x.\langle 1\rangle x \vee \langle \overline{1}\rangle x$.

In order to formally define cycle-free formulas, we first need to extract the sequences of modalities in formulas.

Definition 3.2.1 (Trail of a formula)**.** The *trail of a formula* is the set of sequences of modalities generated by the unfolding of the fixpoints, and is defined as follow:

$$
\begin{aligned}
&trail(p) = trail(\neg p) = trail(\top) = trail(\neg\top) = trail(\neg\langle m\rangle\top) = \epsilon\\
&trail(\langle m\rangle\phi) = \{m \cdot \rho \mid \rho \in trail(\phi)\}\\
&trail(\phi^{\#k}) = trail(\langle 1\rangle\mu x.\phi \vee \langle 2\rangle x)\\
&trail(\phi_1 \wedge \phi_2) = trail(\phi_1 \vee \phi_2) = trail(\phi_1) \cup trail(\phi_2)\\
&trail(\mu x.\phi) = trail(\phi\left[{}^{\mu x.\phi}/_{x}\right])
\end{aligned}
$$

We are now ready to define cycles.

Definition 3.2.2 (Cycles)**.** Given a sequence of modalities ρ, we define the set of cycles in ρ, $cycles(\rho)$, as the set of subsequences $m\overline{m}$ of ρ.

Definition 3.2.3 (Cycle-free formulas)**.** We say a formula ϕ is *cycle-free*, when for every $\rho \in trail(\phi)$, we have that the set $cycles(\rho)$ is finite.

3.2.3 Semantics

Formulas are interpreted as sets of tree nodes. If the formula denotes a non-empty subset of the nodes in a tree, then we say the tree is a *model* for the formula. If there is a model for a formula, then we say the formula is *satisfiable*. When a formula is satisfiable in every model, then we call it a *valid formula*.

Propositions serve as labels for nodes; negation is interpreted as set complement; conjunction and disjunction denote, respectively, set intersection and union; modal formulas denote a one-step multi-directional navigation; least fixpoint is used to perform recursive navigation; and graded formulas impose cardinality constraints on children nodes.

Given a tree K and a valuation V, the formal semantics is defined in Figure 3.6, where:

- V is a valuation function $X \mapsto 2^{\mathcal{N}^K}$;
- $V\left[{}^{\mathcal{N}'}/_{x}\right]$ is written when $V(x) = \mathcal{N}'$, where $\mathcal{N}'$ is a subset of nodes;

Figure 3.6 Semantics of the enriched μ-calculus

$$
\begin{array}{lcl}
[\![p]\!]^K_V & = & \{n \mid \mathcal{L}^K(n) = p\} \\
[\![\neg p]\!]^K_V & = & \{n \mid \mathcal{L}^K(n) \neq p\} \\
[\![\top]\!]^K_V & = & \mathcal{N}^K \\
[\![\neg\top]\!]^K_V & = & \varnothing \\
[\![\neg\langle m\rangle\top]\!]^K_V & = & \{n \mid \mathcal{R}^K(n,m) \text{ is not defined}\} \\
[\![\phi_1 \wedge \phi_2]\!]^K_V & = & [\![\phi_1]\!]^K_V \cap [\![\phi_2]\!]^K_V \\
[\![\phi_1 \vee \phi_2]\!]^K_V & = & [\![\phi_1]\!]^K_V \cup [\![\phi_2]\!]^K_V \\
[\![\langle m\rangle\phi]\!]^K_V & = & \{n \mid \mathcal{R}^K(n,m) \in [\![\phi]\!]^K_V\} \\
[\![\mu\overline{x}.\overline{\phi} \text{ in } \phi]\!]^K_V & = & [\![\phi]\!]^K_{V\left[\overline{\mathcal{N}''}/\overline{x}\right]}, \text{where } \overline{\mathcal{N}''} = \bigcap\left\{\overline{\mathcal{N}'} \mid \overline{[\![\phi]\!]^K_{V\left[\overline{\mathcal{N}'}/\overline{x}\right]}} \subseteq \overline{\mathcal{N}'}\right\} \\
[\![\phi^{\#k}]\!]^K_V & = & \{n \mid |[\![\phi]\!]^K_V \cap \mathit{children}(n)| \# k\}
\end{array}
$$

- $V\left[\overline{\mathcal{N}'}/\overline{x}\right]$ is the n-ary version of $V\left[\mathcal{N}'/x\right]$; and
- for a given node n,

  $\mathit{children}(n) = \{n' \mid \mathcal{R}^K(n,1) = n' \vee [\mathcal{R}^K(n'',2) = n' \wedge n'' \in \mathit{children}(n)]\}$

From the work of [Tarski, 1955], we know that the interpretation of $\mu\overline{x}.\overline{\phi}$ in ϕ corresponds to the least fixpoint.

If the interpretation of two formulas is the same at every tree, we say the formulas are *equivalent*.

An example of the intepretation of formulas is depicted in Figure 3.7.

3.2.4 Encoding of graded modalities

In order to show there is a finite unfolding of every fixpoint, we first encode graded formulas into plain μ-calculus formulas (with no counting operators). Then, we use that fact that there is a finite unfolding for every fixpoint for plain μ-calculus.

The encoding of graded modalities comes at an exponential cost, that is, encoded formulas are exponentially larger than the original ones. Since the size of formulas is the main parameter for the computational cost for satisfiability testing, exponentially larger formulas are exponentially harder to check. Later, in Section 3.3, we introduce a satisfiability algorithm that avoids this blow-up.

We first define the notion of formula size as the number of non logical symbols in the formula. Formal definition of formula size is given in Figure 3.8.

Figure 3.7 Intepretation of an enriched μ-calculus formula

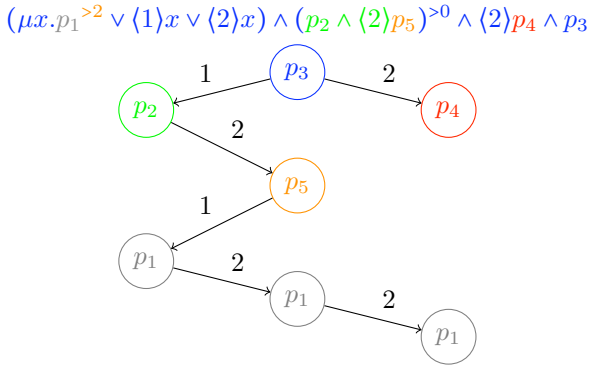


Figure 3.8 Formula size in enriched μ-calculus

$$
\begin{aligned}
&|p| = |\top| = |x| = 1\\
&|\neg\phi| = |\mu x.\phi| = |\phi|\\
&|\langle m\rangle\phi| = |\phi| + 1\\
&|\phi_1 \wedge \phi_2| = |\phi_1 \vee \phi_2| = |\phi_1| + |\phi_2|\\
&|\phi^{\#k}| = |\phi| + k + 1
\end{aligned}
$$

Fixpoint formulas can be used to perform the children navigation involved in graded formulas:

$$\phi^{>0} \equiv \langle 1\rangle \mu x.\phi \vee \langle 2\rangle x$$

In order to mimic the numerical constraints over the children, it is possible to nest the fixpoint formulas.

$$\phi^{>1} \equiv \langle 1\rangle \mu x.(\phi \wedge \langle 2\rangle \mu y.\phi \vee \langle 2\rangle y) \vee \langle 2\rangle x$$

Details about the encoding of graded formulas are given in Figure 3.9.

Lemma 3.2.1 (Encoding graded modalities)**.** *A formula ϕ and its encoding $ch(\phi)$, defined in Figure 3.9, are equivalent, and the size of $ch(\phi)$ is exponentially larger than the size of ϕ.*

Proof. The equivalence is proven by a double induction: one on the structure of the formula; and then, in the case of graded formulas, another one on the numerical constraint.

We now consider the blow-up in the size of ch(ϕ). We proceed also by double induction: on the structure of ϕ, and on the numerical constraints. The only interesting case is when ϕ is a counting formula $\psi^{\#k}$. First consider the

Figure 3.9 Encoding graded formulas in μ-calculus

$\mathrm{ch}(\phi) = \phi$	for $\phi = p, \neg p, x, \top, \neg\top, \neg\langle m\rangle\top$
$\mathrm{ch}(\phi_1 \wedge \phi_2) = \mathrm{ch}(\phi_1) \wedge \mathrm{ch}(\phi_2)$	$\mathrm{ch}(\phi_1 \vee \phi_2) = \mathrm{ch}(\phi_1) \vee \mathrm{ch}(\phi_2)$
$\mathrm{ch}(\langle m\rangle\phi) = \langle m\rangle\mathrm{ch}(\phi)$	$\mathrm{ch}(\mu x.\phi) = \mu x.\mathrm{ch}(\phi)$
$\mathrm{ch}(\phi^{\leq k}) = \neg\mathrm{ch}(\phi^{>k})$	$\mathrm{ch}(\phi^{>k}) = \langle 1\rangle\mu x.(\phi \wedge \mathrm{fsibs}^k(\mathrm{ch}(\phi))) \vee \langle 2\rangle x$
$\mathrm{fsibs}^0(\phi) = \top$	$\mathrm{fsibs}^{k+1}(\phi) = \langle 2\rangle\mu x.(\phi \wedge \mathrm{fsibs}^k(\phi)) \vee \langle 2\rangle x$

case when ψ contains no counting subformulas. Hence, $|\psi^{\#k}| \leq (k+1)|\psi|$. If ψ contains counting subformulas, then $|\psi^{\#k}| \leq (k'+1)^{\overset{l \text{ times}}{\cdots}}(k'+1)|\psi|$, where k' is the greatest numerical constraint occurring in ψ, and l is the greatest level of nesting of counting subformulas. Therefore, $|\psi^{\#k}| \leq (k+1)^{|\psi|}|\psi|$. □

Theorem 3.2.1 (Double exponential satisfiability of enriched μ-calculus)**.** *Formulas of μ-calculus with graded and converse modalities are decidable in double exponential time.*

Proof. [Genevès et al., 2007] showed that the μ-calculus with converse modalities is decidable in single exponential time; then from the exponential blow-up of encoding graded formulas (Lemma 3.2.1), we get the double exponential bound. □

3.2.5 Fixpoint unfolding and normal forms

Since the size of the negation normal form of a formula is not significantly greater than the size of the original formula, then without loss of generality, we consider formulas in negated normal form.

Lemma 3.2.2 (Negation Normal Form)**.** *A formula and its negation normal form are equivalent, furthermore, the negation normal form has a linear size with respect to the original formula.*

Now, we formally introduce the notion of an unfolding.

Definition 3.2.4 (Unfolding)**.** Given a fixpoint formula $\mu x.\phi$, we define its *unfolding* as follows.

$$\begin{aligned} unf^0(\mu x.\phi) &= \mu x.\phi \\ unf^{k+1}(\mu x.\phi) &= \phi\left[{}^{unf^k(\mu x.\phi)}/_x\right] \end{aligned}$$

We are now ready to show there is a finite unfolding for every fixpoint formula.

Lemma 3.2.3 (Finite unfolding)**.** *There is an equivalent finite unfolding for every fixpoint formula, that is,*

$$\left[\!\left[unf^k(\mu x.\phi)\left[{}^{\neg\top}/_{\mu x.\phi}\right]\right]\!\right]^K_V = [\![\mu x.\phi]\!]^K_V\,,$$

for some natural number k.

Proof. From Lemma 3.2.1, we know we can encode graded modalities into plain μ-calculus. [Genevès et al., 2007] showed there is a finite unfolding for plain μ-calculus. □

We can also use the encoding of graded modalities to show there is a guarded normal form for formulas. For a formula in guarded normal form, the variables in fixpoint formulas occur only in the scope of a modal or counting operator.

Theorem 3.2.2 (Guarded Normal Form)**.** *Given a fixpoint formula $\mu x.\phi$, there is an equivalent fixpoint formula $\mu x.\psi$, such that the variable x occurs only in ψ under the scope of a modality or a counting operator.*

Proof. By Lemma 3.2.1, we can encode graded formulas in simple μ-calculus. For a proof for plain μ-calculus (with no counting operators), see for instance [Schneider, 2004]. □

3.2.6 Nominals

We will now show that the logic is expressive enough to restrict the special propositions, called nominals, to occur only once in the models. Since the logic is able to perform multi-directional recursive navigation, from some fixed context node, it is easy to visit all other nodes in the tree. For instance, the formula $EW(\phi)$ holds at every node of every model of ϕ.

$$EW(\phi) \equiv \mu x.\mu y.(\phi \vee \langle 1\rangle y \vee \langle 2\rangle y) \vee \langle \overline{1}\rangle x \vee \langle \overline{2}\rangle x$$

We can now define a global conditional operator, that is, whenever a formula ϕ is true in a model, we select the nodes where ψ holds.

$$\psi \wedge EW(\phi)$$

Nominals are special propositions that only occur once in the model, that is, their interpretation is a singleton. Since we are able to visit all the nodes in a model, then we are also able to express nominals. For this purpose, we first define the auxiliary formulas of Figure 3.10. Then we define a nominal @n as a proposition n which we constrain to hold only once in a model:

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

Figure 3.10 Auxiliary formulas for nominals

descendant(ϕ)	$\equiv$	$\langle 1\rangle \mu x.\phi \vee \langle 1\rangle x \vee \langle 2\rangle x$
following-sibling(ϕ)	$\equiv$	$\mu x.\langle 2\rangle \phi \vee \langle 2\rangle x$
preceding-sibling(ϕ)	$\equiv$	$\mu x.\langle \overline{2}\rangle \phi \vee \langle \overline{2}\rangle x$
descendant-or-self(ϕ)	$\equiv$	$\mu x.\phi \vee \langle 1\rangle \mu y.x \vee \langle 2\rangle y$
ancestor(ϕ)	$\equiv$	$\mu x.\langle \overline{1}\rangle (\phi \vee x) \vee \langle \overline{2}\rangle x$
ancestor-or-self(ϕ)	$\equiv$	$\mu x.\phi \vee \langle \overline{1}\rangle \mu y.x \vee \langle \overline{2}\rangle y$
siblings(ϕ)	$\equiv$	following-sibling(ϕ) $\vee$ preceding-sibling(ϕ)

3.3 Satisfiability Algorithm

In this Section, we present a tableau-based algorithm for checking the satisfiability of formulas. We show that the algorithm is correct: a satisfying tree for a formula is found if and only if the formula is satisfiable. In the last part of the Section, we prove complexity to be exponential.

3.3.1 Overview

The satisfiability algorithm performs two tasks. First, it enumerates all possible satisfying trees of a given formula, and then evaluates the formula against every possible witness model.

In the enumeration of tree models, syntactic characterizations of trees are actually built. At this stage, all possible nodes for the models are considered. These nodes are constructed from a set called the lean. The lean contains the propositions and modal subformulas.

Once we have the nodes, we proceed to iteratively build the trees in a bottom-up manner. First, in the base case, all possible leaves (nodes without children or siblings) are considered. Then, in the iteration step, we consistently connect every possible parent to the subtrees built in previous steps. After each step, we evaluate the consistency of the formula against each tree. In order to ensure termination of this process, the number of identical nodes used to build the trees is bounded.

The algorithm terminates whenever:

- a consistent tree is built, the formula is satisfiable; or
- no more different trees can be built, the formula is unsatisfiable.

3.3.2 Preliminaries

In order to extract the propositional and navigational information contained in the formula, we first consider all subformulas, where fixpoints are expanded only once, and the navigation information of graded formulas is represented by fixpoint formulas performing the children navigation.

Definition 3.3.1 (Fischer-Ladner Closure)**.** For a given formula ϕ, the *Fischer-Ladner closure* $\mathrm{FL}(\phi)$ is defined as the following set.

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

where k is the smallest integer such that $\mathrm{FL}(\phi)_{k+1} = \mathrm{FL}(\phi)_k$, and R^{fl} is defined as follows:

$$R^{\mathrm{fl}}(\phi_1 \wedge \phi_2, \phi_i) \qquad R^{\mathrm{fl}}(\phi_1 \vee \phi_2, \phi_i)$$
$$R^{\mathrm{fl}}(\langle m\rangle\psi, \psi) \qquad R^{\mathrm{fl}}\left(\mu x.\psi, \psi\left[{}^{\mu x.\psi}/_{x}\right]\right) \qquad R^{\mathrm{fl}}(\psi^{\#k}, \langle 1\rangle\mu x.\psi \vee \langle 2\rangle x)$$

for $i = 1, 2$, and $\# \in \{>, \leq\}$.

We now define the set containing the propositional and navigational information composing the nodes.

Definition 3.3.2 (Lean set)**.** Consider a formula ϕ and a proposition σ not occurring in ϕ. We define the *lean* set of ϕ as follows.

$$\mathit{lean}(\phi) = \{p, \langle m\rangle\psi \in \mathrm{FL}(\phi)\} \cup \{\sigma, \langle m\rangle\top\}$$

The syntactic representation of nodes is now defined.

Definition 3.3.3 (ϕ-node)**.** A ϕ-node of a formula ϕ, written n^{ϕ}, is defined as a subset of $\mathit{lean}(\phi)$, such that:

- exactly one non nominal proposition is present;
- when $\langle m\rangle\psi$ occurs, also does $\langle m\rangle\top$; and
- $\langle\overline{1}\rangle\top$ and $\langle\overline{2}\rangle\top$ cannot occur simultaneously.

We will often refer to a ϕ-node simply as a node.

The set of ϕ-nodes is denoted by N^{ϕ}. When the formula under consideration is fixed, we often omit the superscript.

Definition 3.3.4 (ϕtree)**.** A ϕtree is inductively defined as follows:

- an empty tree $\varnothing$ is a tree; and
- a triple $(n^{\phi}, \Gamma_1, \Gamma_2)$ is a tree, provided that Γ_1 and Γ_2 are ϕtrees.

Figure 3.11 Entailment relation for enriched μ-calculus: trees and formulas

$$\frac{}{\Gamma \vdash \top} \qquad \frac{\phi \in n}{(n, \Gamma_1, \Gamma_2) \vdash \phi} \qquad \frac{\phi \notin n}{(n, \Gamma_1, \Gamma_2) \vdash \neg\phi} \qquad \frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}$$

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi} \qquad \frac{\Gamma \vdash \phi\left[{}^{\mu x.\phi}/_{x}\right]}{\Gamma \vdash \mu x.\phi} \qquad \frac{\Gamma_1 \vdash \phi^{k'} \quad k' \# k}{(n, \Gamma_1, \Gamma_2) \vdash \phi^{\# k}}$$

Figure 3.12 Auxiliary entailment relation: counting on siblings

$$\frac{(n, \varnothing, \varnothing) \vdash \neg\phi}{(n, \varnothing, \varnothing) \vdash \phi^0} \qquad \frac{(n, \varnothing, \varnothing) \vdash \phi}{(n, \varnothing, \varnothing) \vdash \phi^1}$$

$$\frac{(n, \Gamma_1, \Gamma_2) \vdash \neg\phi \quad \Gamma_2 \vdash \phi^k}{(n, \Gamma_1, \Gamma_2) \vdash \phi^k} \qquad \frac{(n, \Gamma_1, \Gamma_2) \vdash \phi \quad \Gamma_2 \vdash \phi^{k-1}}{(n, \Gamma_1, \Gamma_2) \vdash \phi^k}$$

When clear from the context, we refer to ϕtrees simply as trees.

Definition 3.3.5. The set of subtrees of a given tree is defined as follows:

- the set of subtrees of the empty tree is the empty set.
- the set of subtrees of a tree (n, Γ_1, Γ_2) is the union of $\{(n_1, \Gamma_1, \Gamma_2)\}$ with the subtrees of Γ_1 and Γ_2.

We now define the notion of consistent trees. To this end, we define an *entailment relation* between trees and formulas as shown in Figure 3.11. In Figure 3.12, we define an auxiliary entailment relation in charge of counting sibling nodes, where $\neg\phi$ means the negation normal form of ϕ.

Definition 3.3.6 (Modal consistency)**.** Given a formula ϕ, two trees Γ_1, Γ_2, and a node n_0, we say the tree $(n_0, \Gamma_1, \Gamma_2)$ is *modally consistent*, written $\Delta(n_0, \Gamma_1, \Gamma_2)$, if and only if, for $i = 1, 2$, we have that

$$\forall \langle i \rangle \psi \in \mathit{lean}(\phi), \langle i \rangle \phi \in n_0 \iff \Gamma_i \vdash \phi$$
$$\forall \langle \bar{i} \rangle \psi \in \mathit{lean}(\phi), \langle \bar{i} \rangle \phi \in n_i \iff (n_0, \Gamma_1, \Gamma_2) \vdash \phi$$

We conclude the preliminaries by introducing some final notations. The root of a tree is defined as follows:

$$\begin{array}{lcl} \mathit{root}(\varnothing) & = & \varnothing \\ \mathit{root}((n, \Gamma_1, \Gamma_2)) & = & n \end{array}$$

A tree Γ *satisfies* a formula ϕ if there is a subtree Γ' in Γ such that

- Γ' entails ϕ, that is $\Gamma' \vdash \phi$; and

- neither $\langle\overline{1}\rangle\top$ nor $\langle\overline{2}\rangle\phi$ occur in $root(\Gamma)$.

We write $\Gamma \Vdash \phi$ when Γ satisfies ϕ. Given a set of trees ST, we write $ST \Vdash \phi$ when there is a tree in ST satisfying ϕ.

3.3.3 The Algorithm

We now present the satisfiability-checking algorithm. Candidate trees are consistently built in a bottom-up manner. At each step, the algorithm verifies if there is a tree satisfying the formula. If such a satisfying tree is found, the algorithm returns *true*. The set of nodes is finite, and the number of occurrences of children nodes is bounded by $maxK(\phi)$ (defined in Figure 3.13). Hence, the number of steps in the algorithm is finite. If no satisfying tree is found, then the algorithm returns *false*.

Algorithm 1 Enriched μ-Calculus: Check Satisfiability of ϕ

$ST \leftarrow \varnothing$
repeat
  $AUX \leftarrow \{(n, \Gamma_1, \Gamma_2) \mid$ {we extend the trees}
    $\text{maxch}(n, \Gamma_2) \leq maxK(\phi)$ {with an available node}
    $\Delta(n, \Gamma_1, \Gamma_2)$ {checking consistency}
    where $\Gamma_i \in ST \cup \{\varnothing\}\}$
  if $AUX \subseteq ST$ **then**
    return *false* {no new tree was built}
  end if
  $ST \leftarrow ST \cup AUX$
until $ST \Vdash \phi$
return *true*

To bound the size of the trees that are built, we restrict the number of identical children nodes by $maxK(\phi)$, defined in Figure 3.13. The function maxch is in charge of counting the identical children nodes (also defined in Figure 3.13).

Consider for instance the formula $\phi = p_1 \wedge p_2^{\geq 2}$. The computed lean is as follows, where $\psi = \mu x.p_2 \vee \langle 2\rangle x$:

$$\{p_1, p_2, p_3, \langle 1\rangle\top, \langle 2\rangle\top, \langle\overline{1}\rangle\top, \langle\overline{2}\rangle\top, \langle 1\rangle\psi, \langle 2\rangle\psi\}$$

Labels other than p_1 and p_2 are represented by p_3. The bound on children nodes $maxK(\phi)$ is 3.

After the first step, ST consists of the following trees, where $i = 1, 2, 3$ and $j = 1, 2$.

$$\{(\{p_i\}, \varnothing, \varnothing), (\{p_i, \langle\overline{j}\rangle\top\}, \varnothing, \varnothing)\}$$

At this point the three finished trees are tested and found not to satisfy ϕ.

Many trees are created after the second iteration, but the one of interest is the following:

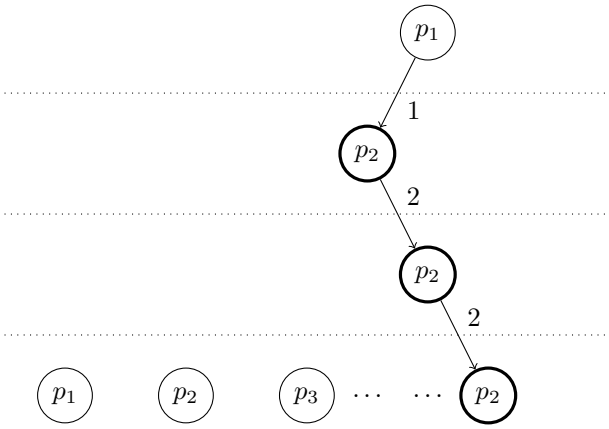
$$\Gamma_0 = (\{p_2, \langle 2\rangle\psi, \langle 2\rangle\top, \langle\overline{2}\rangle\top\}, \varnothing, (\{p_2, \langle\overline{2}\rangle\top\}, \varnothing, \varnothing))$$

Figure 3.13 Enriched μ-calculus: occurrences bound functions

$$maxK(p) = maxK(\neg p) = maxK(\neg\langle m\rangle\top) = maxK(\top) = maxK(x) = 0$$
$$maxK(\phi_1 \wedge \phi_2) = maxK(\phi_1 \vee \phi_2) = maxK(\phi_1) + maxK(\phi_2)$$
$$maxK(\langle m\rangle\phi) = maxK(\mu x.\phi) = maxK(\phi)$$
$$maxK(\phi^{\#k}) = k + 1 + maxK(\phi)$$

$$\begin{array}{lcl} \text{maxch}(n, (n, \Gamma_1, \Gamma_2)) & = & 1 + \text{maxch}(n, \Gamma_2) \\ \text{maxch}(n, (n', \Gamma_1, \Gamma_2)) & = & \text{maxch}(n, \Gamma_2), \text{where } n \neq n' \\ \text{maxch}(n, \varnothing) & = & 1 \end{array}$$

Figure 3.14 Checking $\phi = p_1 \wedge p_2^{\geq 2}$



The third iteration yields the following tree.

$$\Gamma_1 = (\{p_2, \langle 2\rangle\psi, \langle 2\rangle\top, \langle\overline{1}\rangle\top\}, \varnothing, \Gamma_0)$$

We can conclude at the fourth iteration when we find the tree satisfying ϕ.

$$(\{p_1, \langle 1\rangle\psi, \langle 1\rangle\top\}, \Gamma_1, \varnothing)$$

As there is no repetition in the children nodes, the bound $maxK(\phi)$ is satisfied. Figure 3.14 depicts a graphical representation of the example, where counted nodes are drawn as thick circles.

3.4 Correctness and Complexity

Termination of the algorithm is ensured because the number of nodes is finite, and there is a bound on the number of identical nodes permitted in the trees.

We now show that the algorithm is sound and complete, that is, a formula is satisfiable if and only if the algorithm returns *true*.

The complexity of the algorithm is analysed in the last part of the Section.

3.4.1 Soundness

If the algorithm terminates with a candidate tree satisfying a given formula, that is, the algorithm returns *true*, we need to show that the formula is satisfiable. To this end, we first define a tree structure out of a ϕtree.

Definition 3.4.1. Given a ϕtree Γ, we define the tree structure $K(\Gamma) = (\mathcal{N}, \mathcal{R}, \mathcal{L})$, such that:

- $\mathcal{N} = \{n \mid n \text{ is a } \phi\text{node in } \Gamma\}$;
- for every subtree (n, Γ_1, Γ_2) of Γ, if $\Gamma_i \neq \varnothing$, then $\mathcal{R}(n, i) = root(\Gamma_i)$; and
- if a proposition $p \in n \in \mathcal{N}$, then $\mathcal{L}(n) = p$.

We now show that $K(\Gamma)$ satisfies ϕ.

Theorem 3.4.1 (Soundness)**.** *Given a formula ϕ, if the satisfiability algorithm returns* 1*, that is, $\Gamma \Vdash \phi$ where $\Gamma \in ST$, then $K(\Gamma)$ satisfies ϕ.*

Proof. By induction on the structure of ϕ.

When ϕ is either p, $\top$, $\neg p$ or $\neg\langle m\rangle\top$, then the algorithm stops at the first step. Hence, only leaf nodes are considered. From definition of $K(\phi)$, it is easy to see that ϕ is satisfiable.

The cases for disjunction and conjunction are also straightforward. If it is disjunction $\psi_1 \vee \psi_2$, then by induction at least one ψ_i ($i \in \{1, 2\}$) is satisfied by $K(\Gamma)$. In the conjunction case $\psi_1 \wedge \psi_2$, we get also by induction that both ψ_i ($i = 1, 2$) are satisfied by $K(\Gamma)$.

Consider now when ϕ is a modal formula $\langle m\rangle\psi$. Then we know that there is a subree Γ_1 of Γ, such that $\Gamma_1 \vdash \langle m\rangle\psi$. By construction of Γ, there is another subtree Γ_2 of Γ, such that:

- $\Gamma_2 \vdash \psi$; and
- if $m = 1$, then we have that
  - $\Delta_1(root(\Gamma_1), \Gamma_2, \Gamma_3)$, or
  - $\Delta_1(root(\Gamma_2), \Gamma_1, \Gamma_3)$;
- if $m = 2$, then we have that
  - $\Delta_2(root(\Gamma_1), \Gamma_3, \Gamma_2)$, or
  - $\Delta_2(root(\Gamma_2), \Gamma_3, \Gamma_1)$;

From $\Gamma_2 \vdash \psi$ and by induction, we get that $K(\Gamma)$ satisfies ψ. Then, consistently with Δ_m, we obtain that $\mathcal{R}(root(\Gamma_1), m) = root(\Gamma_2)$. Hence, $K(\Gamma)$ also satisfies $\langle m \rangle \psi$.

If ϕ is a fixpoint formula $\mu x.\psi$, then we proceed by induction on k, where k is the smallest number such that $unf^k(\mu x.\psi)\left[^{\neg\top}/_{\mu x.\psi}\right]$ is equivalent to $\mu x.\psi$ (Lemma 3.2.3). The base case ($k = 1$) is easy, since by induction on the structure, $unf^1(\mu x.\psi) = \psi\left[^{\neg\top}/_{x}\right]$ is also satisfied by $K(\Gamma)$. We consider now the induction step. Note that $\mu x.\psi$ is equivalent to $\psi\left[^{\mu x.\psi}/_{x}\right]$, hence the equivalent finite unfolding of $\psi\left[^{\mu x.\psi}/_{x}\right]$ is $unf^{k-1}(\mu x.\psi)\left[^{\neg\top}/_{\mu x.\psi}\right]$. By inductive hypothesis on k, $unf^{k-1}(\mu x.\psi)\left[^{\neg\top}/_{\mu x.\psi}\right]$ is satisfied by $K(\Gamma)$, then by induction $unf^k(\mu x.\psi)\left[^{\neg\top}/_{\mu x.\psi}\right]$ is also satisfied. Therefore, we get $K(\Gamma)$ that satisfies $\mu x.\psi$.

We now consider the cases when ϕ is a graded formula $\psi^{>k}$ or $\psi^{\leq k}$. Given that $\Gamma \vdash \psi^{\#k}$, we know by rules in Figures 3.11 and 3.12, that there is sequence of subtrees $\Gamma_0, \ldots, \Gamma_{k'}$ in Γ, such that for $i = 1, \ldots, k'$, we have that:

- $\Gamma_0 \vdash \psi^{\#k}$ and $\Gamma_i \vdash \psi$;
- the roots of Γ_i are children of the root of Γ_0; and
- $k' \# k$.

Then by induction and construction of Γ, we get that there is a sequence of nodes $n_0, \ldots, n_{k'}$ in $K(\phi)$ such that n_0 is the parent of n_i ($i = 1, \ldots, k'$) and ψ holds a each n_i. Therefore, since $k' \# k$, we get that n_0 satisfies $\psi^{\#k}$. □

3.4.2 Completeness

We now show that given a tree structure K satisfying a formula ϕ, the algorithm constructs a satisfying ϕtree for ϕ, that is, the algorithm returns *true*.

The proof is divided in two main steps. First we build a ϕtree out of the satisfying K structure, and then we show that the ϕtree also satisfies ϕ. The second step consists in showing that the algorithm can actually construct such a satisfying ϕtree.

We first define a lean labelled ϕtree homomorphic to K.

Definition 3.4.2. Given a tree structure K satisfying a formula ϕ, we define a ϕtree $\Gamma(K)$ as follows:

- for every node n in K, there is a corresponding node n' in $\Gamma(K)$, such that for every ψ in $lean(\phi)$, if $n \in [\![\psi]\!]^K_{\varnothing}$, then $\psi \in n'$; and
- for every node n_0 in K, if $\mathcal{R}^K(n_0, 1) = n_1$ and $\mathcal{R}^K(n_0, 2) = n_2$, then $(n'_0, \Gamma_1, \Gamma_2)$ is a subtree of $\Gamma(K)$, where $root(\Gamma_i) = n'_i$ ($i = 1, 2$) and n'_1, n'_2 and n'_0 are the corresponding nodes, defined in the previous step, of n_1, n_2 and n_0. When $\mathcal{R}^K(n_0, i)$ ($i \in \{1, 2\}$) is not defined, then $\Gamma_i = \varnothing$.

We now show that $\Gamma(K)$ satisfies the original formula.

Lemma 3.4.1. *If a tree structure K satisfies a formula ϕ, then the ϕtree $\Gamma(\phi)$ also does, that is, $\Gamma(\phi) \Vdash \phi$.*

Proof. We consider the smallest tree structure satisfying ϕ, and we proceed by induction on the structure of ϕ.

The base cases are when ϕ is p, $\top$, $\neg p$ and $\neg\langle m\rangle\phi$. The smallest tree structure satisfying ϕ is a leaf. In the case of p, since p is in the lean, then the leaf in $\Gamma(K)$ satisfies p. If ϕ is $\top$, then any leaf is fine. As for the negated cases $\neg p$ and $\neg\langle m\rangle\phi$, none of them belongs to the lean, so the leaf satisfies them.

Conjunction and disjunction cases are straightforward. By induction each/a member of the conjunction/disjunction is satisfied by $\Gamma(K)$, and so ϕ is also satisfied.

We now refer to the case when ϕ is a modal formula $\langle m\rangle\psi$ and holds at node n. Since $\langle m\rangle\psi$ is in the lean, and according to the construction of $\Gamma(K)$, then the corresponding node n' in $\Gamma(K)$ of n contains $\langle m\rangle\psi$, and hence $\Gamma(K)$ satisfies $\langle m\rangle\psi$.

When ϕ is a fixpoint formula $\mu x.\psi$, we consider the equivalent formula $\psi\left[{}^{\mu x.\psi}/_{x}\right]$. Then we proceed by another induction on the structure of ψ. All the cases, but the fixpoint, are similar to the first structural induction applied in this proof. Fixpoints never occur in this induction step: recall that variables in fixpoint formulas occur only under the scope of modalities or counting operators (Theorem 3.2.2).

Consider now the case where $\psi^{\#k}$ holds at n_0. Then we know ψ holds in the children nodes $n_1, \ldots, n_{k'}$ of n_0 such that $k'\#k$, and for every child node n' of n_0 different from n_i, we have that ψ does not hold there. Then by induction we get that $\Gamma_i \vdash \psi$ $(i = 1, \ldots, k')$, where $root(\Gamma_i)$ are the corresponding nodes in $\Gamma(K)$ of n_i. Therefore, $\Gamma(K) = (n', \Gamma_1, \Gamma_2)$ satisfies $\psi^{\#k}$, where n'_0 is the corresponding node of n_0 and $\Delta(n'_0, \Gamma_1, \Gamma_2)$ holds. □

We next show that the $\Gamma(K)$ can actually be built by the algorithm. We closely follow the proof of [Genevès et al., 2007], for the μ-calculus with no graded modalities, with a crucial exception: we need to make sure there are enough instances of each child node. Indeed, in [Genevès et al., 2007], the algorithm uses a node (a subset of the lean) at most once. This yields a simple halting condition for the algorithm to conclude that the formula is unsatisfiable. However, in the presence of graded formulas, a given child node can occur more than once. To maintain the termination of the algorithm, we put a bound on the number of identical children nodes that may be needed by $maxK(\phi)$, as defined in Figure 3.13. We need also to show that this bound is sufficient to build a tree for any satisfiable formula.

[Kupferman et al., 2002] first showed that $maxK(\phi)$ is a bound for the number of children nodes for the graded μ-calculus with no converse modalities: "if a formula ϕ is satisfiable (by a Kripke graph), then there is a tree model (possibly infinite) such that each node has at most $l(k+1)$ children, where l is the number of subformulas $\psi^{>k'}$ and k is the greatest numerical constraint occurring in such graded subformulas". This result was extended for graded μ-

calculus with converse, called fully graded μ-calculus, by [Bonatti et al., 2006]. We now show this result also applies in the finite tree setting.

Theorem 3.4.2. *If a formula ϕ is satisfiable, then there is a model of ϕ such that each node hast at most $l(k+1)$ children, where l is the number of subformulas $\psi^{>k'}$ and k is the greatest numerical constraint occurring in such graded subformulas".*

Proof. We proceed by contradiction. It is assumed that ϕ is satisfiable and there is no tree where each node has at most $l(k+1)$ nodes. Consider K_0 to be such tree. We now build a tree K_1 isomorphic to K_0 such that each node in K_1 contains the lean formulas satisfied by their corresponding node in K_0. It is immediate that K_1 also satisfies ϕ.

We then build a tree K_2 from K_1 such that all "unnecessary" children in K_1 are removed, that is, where each node in K_2 has at most $l(k+1)$ children. Let n be the node in $\mathcal{N}^{K_1}$ that satisfies ϕ, we then define $\mathcal{N}^{K_2} = \mathcal{N}(\phi, n)$, where:

- if ϕ is a atomic formula $p, \top, \neg\top, \neg\langle m\rangle\top$, then $\mathcal{N}(\phi, n) = \{n\}$;
- $\mathcal{N}(\phi_1 \wedge \phi_2, n) = \mathcal{N}(\phi_1, n) \cap \mathcal{N}(\phi_2, n)$;
- $\mathcal{N}(\phi_1 \vee \phi_2, n) = \mathcal{N}(\phi_1, n) \cup \mathcal{N}(\phi_2, n)$;
- $\mathcal{N}(\langle m\rangle\psi, n) = \mathcal{N}(\psi, n')$, such that $m \in \{2, \overline{2}\}$, n' satisfies ψ and it is a following (preceding) sibling of n according to K_1;
- $\mathcal{N}(\langle m\rangle\psi, n) = \mathcal{N}(\psi, n')$, such that $\mathcal{R}^{K_1}(n, m) = n'$ and $m \in \{1, \overline{1}\}$;
- $\mathcal{N}(\mu x.\psi, n) = \mathcal{N}(unf^k(\mu x.\psi), n)$, such that $unf^k(\mu x.\psi)$ is the smallest unfolding of $\mu x.\psi$;
- $\mathcal{N}(\psi^{>k}, n) = \{n\} \cup \mathcal{N}(n_i, \psi)$, such that $i = 1, \ldots, k+1$, n is the parent of n_i and n_i satisfies ψ; and
- $\mathcal{N}(\psi^{\leq k}, n) = \{n\} \cup \mathcal{N}(n_i, \psi)$, such that $i = 1, \ldots, k' \leq k$, n is the parent of n_i and n_i satisfies ψ.

We now define $\mathcal{R}^{K_2} = \mathcal{R}(\phi, n)$, where:

- if ϕ is a atomic formula, then $\mathcal{R}(\phi, n) = \varnothing$;
- $\mathcal{R}(\phi_1 \wedge \phi_2, n) = \mathcal{R}(\phi_1, n) \cup \mathcal{R}(\phi_2, n)$;
- $\mathcal{R}(\phi_1 \vee \phi_2, n) = \mathcal{R}(\phi_i, n)$ for $i \in \{1, 2\}$, or $\mathcal{R}(\phi_1 \wedge \phi_2, n) = \mathcal{R}(\phi_1, n) \cup \mathcal{R}(\phi_2, n)$;
- $\mathcal{R}(\langle m\rangle\psi, n) = \mathcal{R}(\psi, n') \cup \{(n, m, n')\}$, such that $m = \{1, \overline{1}\}$ and $\mathcal{R}^{K_1}(n, m) = n'$;

- $\mathcal{R}(\langle m\rangle\psi, n) = \mathcal{R}(\psi, n') \cup \{(n, m, n')\}$, such that $m = \{2, \overline{2}\}$ and $\mathcal{R}^{K_1}(n, m) = n'$, or in case $\mathcal{R}^{K_1}(n, m) \neq n'$, then n' is a following (preceding) sibling of n according to K_1 and $\Delta_m(n, n')$;
- $\mathcal{R}(\mu x.\psi, n) = \mathcal{R}(unf^k(\mu x.\psi), n)$, such that $unf^k(\mu x.\psi)$ is the smallest unfolding of $\mu x.\psi$;
- $\mathcal{R}(\psi^{>k}, n) = \bigcup_{i=1}^{k+1} \mathcal{R}(\psi, n_i) \cup (n, 1, n_1) \cup \bigcup_{j=2}^{k} (n_j, 2, n_{j+1})$, such that n is the parent of n_i according to K_1, $\Delta_1(n, n_1)$ and $\Delta_2(n_i, n_{i+1})$ $(i = 2, \ldots, k)$; and
- $\mathcal{R}(\psi^{\leq k}, n) = \bigcup_{i=1}^{k'} \mathcal{R}(\psi, n_i) \cup (n, 1, n_1) \cup \bigcup_{j=2}^{k'-1} (n_j, 2, n_{j+1})$, such that $k' \leq k$, n is the parent of n_i according to K_1, $\Delta_1(n, n_1)$ and $\Delta_2(n_i, n_{i+1})$ $(i = 2, \ldots, k-1)$.

It is now shown by structural induction on ϕ that K_2 satisfies ϕ, and each node in K_2 has no more than $l(k+1)$ children. This is immediatte from the construction of K_2. □

We are now ready to show the completeness.

Theorem 3.4.3 (Completeness)**.** *If a formula is satisfiable, then a satisfying tree is built by the algorithm.*

Proof. We consider the smallest tree structure K satisfying a formula ϕ, and we proceed by induction on the height of K.

The base case is straightforward, since K is a leaf, and all leaves are produced by the algorithm in the first step.

For the induction step, we consider a tree structure K with height $k+1$. Consider the two main subtrees K_1 and K_2 of K, such that $\mathcal{R}^K(n_0, i) = n_i$ $(i = 1, 2)$, where n_i is the root of K_i and n_0 is the root of K. Since both K_i have a height less or equal than k, then by induction, we know the trees $\Gamma(K_i)$ have been produced by the algorithm in ST. It is now easy to see that $\mathcal{R}^K(n_0, i) = n_i$ implies $\Delta(n_0', \Gamma(K_1), \Gamma(K_2))$, where n_0' is the corresponding node in $\Gamma(K)$ of n_0. That n_0' is available, that is, $\text{maxch}(n_0', \Gamma(K_2)) \leq maxK(\phi)$, comes from Theorem 3.4.2. Therefore $\Gamma(K)$ is built by the algorithm. □

3.4.3 Complexity

We now show that the time complexity of the satisfiability algorithm is exponential in the formula size. We show this result in two steps: we first show that the lean size is linear with respect to the formula size; then we show that the algorithm has a single exponential complexity with relation to the lean size.

Lemma 3.4.2. *The lean size is linear in terms of the original formula size.*

Proof Sketch. By the size of the lean we mean the number of elements it contains; we do not care about the size of its elements.

[Genevès et al., 2007] showed the lean size of μ-calculus formulas, that does not include graded formulas, is linear w.r.t. formula size.

Graded formulas $\psi^{\#k}$ only introduce two extra formulas to the lean: $\langle 1\rangle\xi$ and $\langle 2\rangle\xi$, where ξ is $\langle 1\rangle\mu x.\psi \vee \langle 2\rangle x$. □

Theorem 3.4.4. *The satisfiability algorithm for the logic is decidable in time $2^{O(n)}$, where n is the size of the formula.*

Proof Sketch. By Lemma 3.4.2, we know the size of the lean is linear in terms of the formula size. Now, we will show the algorithm complexity is exponential w.r.t. to the size of the lean.

Since a node is defined as a subset of the lean, then the size of the set containing all the nodes is bounded by 2^k, where k is the lean size. Now, the number of identical nodes that may occur in every tree is bounded by $maxK(\phi) \leq l(m+1)$, where l is the number of graded formulas occurring in ϕ, and m is the greatest numerical constraint occurring in such graded formulas. Hence, the number of steps of the algorithm is bounded by $2^k * l * m$.

Three operations are performed at each step in the loop cycle of the algorithm: the definition of AUX; a subset inclusion $AUX \subset ST$; and a set union $ST \cup AUX$.

In order to determine the cost of AUX, we need to examine the cost of its three main operations: the traversals to form the triples (n, Γ_1, Γ_2); the cost of computing the function maxch; and the cost of the modal compatibility relation Δ. Since the number of different nodes is exponential, and the number of different subtrees too, the maximum number of newly formed trees (triples) at each step also has an exponential bound. The function maxch performs a single traversal of the tree which is also exponential. In the definition of the relation Δ, the entailment relation performs a traversal of the tree to verify each graded formula. Hence, the cost of Δ is also exponentially bounded.

The set inclusion $AUX \subseteq ST$ and the set union $ST \cup AUX$ are linear operations in terms of exponentially sized sets.

The stop condition of the algorithm is checked by the satisfaction relation $\Vdash$. It involves traversals parametrized by the number of trees, the number of nodes in each tree, and the number of traversals for the entailment relation of each graded formula. Hence, the upper bound for the stop condition is $(2^k * l * m)^3$.

Therefore, the total time complexity of the algorithm is bounded by $(2^k * l * m)^{k'}$, for some constant k'. □

3.5 Application to Regular Paths

[Clark and DeRose, 1999] introduced XPath as part of the W3C XSLT transformation language. XPath is a language for selecting nodes and computing values from an XML document. Since it became a standard, XPath has been included as part of several other standards, in particular it forms the "navigation subset" of the XQuery language.

Figure 3.15 Syntax of Core XPath expressions

Axis	:=	self | child | parent | descendant | ancestor |
		following-sibling | preceding-sibling | following | preceding
NameTest	:=	QName | $*$
Step	:=	Axis :: NameTest
PathExpr	:=	PathExpr/PathExpr | PathExpr[Qualifier] | Step
Qualifier	:=	PathExpr | Qualifier and Qualifier | Qualifier or Qualifier |
		not Qualifier | CountExpr
StepCh	:=	child :: NameTest
CountExpr	:=	$count$(StepCh)$\#k$ | $count$(StepCh[Qualifier])$\#k$
#	:=	$\leq \mid > \mid =$
XPathExpr	:=	PathExpr | /PathExpr | XPathExpr union PathExpr |
		XPathExpr intersect PathExpr |
		XPathExpr except PathExpr

In the first part of this Section, we give a formal presentation of an extension of what is known as Core XPath. The extension consists in considering XPath expressions with numerical constraints on the number of the selected nodes.

[Genevès et al., 2007] have already shown there is a linear translation of Core XPath expressions (without counting) into the μ-calculus. We show in this Section how to extend this translation to XPath counting expressions with graded formulas of the enriched μ-calculus.

3.5.1 XPath expressions

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

/company/personnel/employee

first navigates from the root of a document through the top-level "company" node to its "personnel" children nodes and from there, to "employee" children 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.

Each step of the navigation of XPath expressions can perform more sophisticated navigation to select nodes. For instance, the selection of "descendant" nodes requires a recursive navigation. XPath expressions are also able to navigate upward and recursively, as represented by the "parent" and "ancestor" axes.

Figure 3.16 Interpretation of Core XPath expressions

$$
\begin{array}{lcl}
[\![\text{Axis} :: \text{NameTest}]\!] & = & \{(n_1, n_2) \in \mathcal{N} \times \mathcal{N} \mid n_2 \text{ is an Axis of } n_1, \\
 & & \text{and } n_2 \text{ is labeled by NameTest}\} \\
[\![/P]\!] & = & \{(n_1, n_2) \in \mathcal{N} \times \mathcal{N} \mid n_2 \text{ is the root, } n_2 \in [\![P]\!]\} \\
[\![P_1/P_2]\!] & = & [\![P_1]\!] \circ [\![P_2]\!] \\
[\![P_1 \text{ union } P_2]\!] & = & [\![P_1]\!] \cup [\![P_2]\!] \\
[\![P_1 \text{ intersect } P_2]\!] & = & [\![P_1]\!] \cap [\![P_2]\!] \\
[\![P_1 \text{ except } P_2]\!] & = & [\![P_1]\!] \setminus [\![P_2]\!] \\
[\![P[Q]]\!] & = & \{(n_1, n_2) \in [\![P]\!] \mid n_2 \in [\![Q]\!]^q\} \\
[\![P]\!]^q & = & \{n_1 \mid \exists n_2 : (n_1, n_2) \in [\![P]\!]\} \\
[\![count(P)\#k]\!]^q & = & \{n_1 \mid |\{n_2 \mid (n_1, n_2) \in [\![P]\!]\}|\#k\} \\
[\![Q_1 \text{ and } Q_2]\!]^q & = & [\![Q_1]\!]^q \cap [\![Q_2]\!]^q \\
[\![Q_1 \text{ or } Q_2]\!]^q & = & [\![Q_1]\!]^q \cup [\![Q_2]\!]^q \\
[\![\text{not } Q]\!]^q & = & \mathcal{N} \setminus [\![Q]\!]^q
\end{array}
$$

One of the most powerful features of XPath expressions is the ability to "filter" the selection of nodes at each navigation step. For instance, the expression

/company[descendant::employee/name/Pierre]/name

selects company names that have at least one employee named Pierre. Often, it is of interest to have more general numerical constraints than "at least one". For instance,

/company/personnel[count(employee)> 10000]/name

which selects the name of companies with more than 10000 employees.

The syntax and semantics of Core XPath, extended with counting constraints on children nodes, are respectively given in Figure 3.15 and Figure 3.16. XPath expressions are interpreted as pairs of nodes in a tree structure. The considered XPath fragment allows absolute and relative paths, path union, intersection, composition, as well as node test and qualifiers with counting operators on children nodes, conjunction, disjunction, negation, and path navigation. Furthermore, it supports all XPath axes allowing multi-directional and recursive navigation.

3.5.2 XPath embedding

We will now show we can consistently encode XPath expressions with counting constraints into the μ-calculus with graded modalities. Furthermore, we will

Figure 3.17 Logic embedding of Axis

$$
\begin{array}{lcl}
\langle \text{self} \rangle_\chi & = & \chi \\
\langle \text{child} \rangle_\chi & = & \mu x.\langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle x \\
\langle \text{following-sibling} \rangle_\chi & = & \mu x.\langle \overline{2} \rangle \chi \vee \langle \overline{2} \rangle x \\
\langle \text{preceding-sibling} \rangle_\chi & = & \mu x.\langle 2 \rangle \chi \vee \langle 2 \rangle x \\
\langle \text{parent} \rangle_\chi & = & \langle 1 \rangle \mu x.\chi \vee \langle 2 \rangle x \\
\langle \text{descendant} \rangle_\chi & = & \mu x.\langle \overline{1} \rangle (\chi \vee x) \vee \langle \overline{2} \rangle x \\
\langle \text{descendant-or-self} \rangle_\chi & = & \mu y.\chi \vee \mu x.\langle \overline{1} \rangle (y \vee x) \vee \langle \overline{2} \rangle x \\
\langle \text{ancestor} \rangle_\chi & = & \langle 1 \rangle \mu x.\chi \vee \langle 1 \rangle x \vee \langle 2 \rangle x \\
\langle \text{ancestor-or-self} \rangle_\chi & = & \mu x.\chi \vee \langle 1 \rangle \mu y.x \vee \langle 2 \rangle y \\
\langle \text{following} \rangle_\chi & = & \langle \text{descendant-or-self} \rangle_{\langle \text{following-sibling} \rangle_{\langle \text{ancestor-or-self} \rangle_\chi}} \\
\langle \text{preceding} \rangle_\chi & = & \langle \text{descendant-or-self} \rangle_{\langle \text{preceding-sibling} \rangle_{\langle \text{ancestor-or-self} \rangle_\chi}}
\end{array}
$$

show that the encoding is linear. That is, the size of the translation of the XPath expressions into the logic is linear with respect to the size of the original XPath expression.

Figure 3.17 shows how XPath navigation is translated into the μ-calculus.

Logical formulas are also able to capture step navigation in XPath expressions. For example, consider the following expression with its corresponding translation.

$$
\begin{array}{l}
\textbf{child::}\mathbf{p_1}/\text{descendant} :: p_2 \\
p_2 \wedge \mu y.(\mathbf{p_1} \wedge \mu \mathbf{x}.\langle \overline{\mathbf{1}} \rangle \top \vee \langle \overline{\mathbf{2}} \rangle \mathbf{x}) \vee \mu x.\langle \overline{1} \rangle (y \vee x) \vee \langle \overline{2} \rangle x
\end{array}
$$

In contrast with the interpretation of paths outside qualifiers, which actually selects nodes reachable by the path, paths inside qualifiers are interpreted as boolean functions, that is, their interpretation requires to navigate through the path, in order to test the existence of nodes, without changing the focus nodes achieved before the qualifier. In order to maintain such focus, we interpret the paths inside qualifiers in an inverse manner. In order to illustrate how the translation works, consider the following expression and its corresponding translation.

$$
\begin{array}{l}
\textbf{child::}\mathbf{p_1}[\text{child} :: p_2] \\
\mathbf{p_1} \wedge \mu \mathbf{x}.\langle \overline{\mathbf{1}} \rangle \top \vee \langle \overline{\mathbf{2}} \rangle \mathbf{x} \wedge \langle 1 \rangle \mu x.p_2 \vee \langle 2 \rangle x
\end{array}
$$

From this example, we can see that forward and backward navigation are crucial to succinctly capture XPath expressions.

Figure 3.18 Logic embedding of XPath expressions

$\langle * \rangle$	=	$\top$
$\langle \text{QName} \rangle$	=	QName
$\langle \text{Axis} :: \text{NameTest} \rangle^p_\chi$	=	$\langle \text{Axis} \rangle_\chi \wedge \langle \text{NameTest} \rangle$
$\langle P_1/P_2 \rangle^p_\chi$	=	$\langle P_2 \rangle^p_{\langle P_1 \rangle^p_\chi}$
$\langle P[Q] \rangle^p_\chi$	=	$\langle P \rangle^p_\chi \wedge \langle Q \rangle^q_\top$
$\langle /P \rangle_\chi$	=	$\langle P \rangle^p_{\mu x.(\neg\langle\overline{1}\rangle\top \vee \langle\overline{2}\rangle x) \wedge \mu y.(\chi \wedge \textcircled{s}) \vee \langle 1\rangle y \vee \langle 2\rangle y}$
$\langle P \rangle_\chi$	=	$\langle P \rangle^p_{\chi \wedge \textcircled{s}}$
$\langle P_1 \text{ union } P_2 \rangle_\chi$	=	$\langle P_1 \rangle_\chi \vee \langle P_2 \rangle_\chi$
$\langle P_1 \text{ intersect } P_2 \rangle_\chi$	=	$\langle P_1 \rangle_\chi \wedge \langle P_2 \rangle_\chi$
$\langle P_1 \text{ except } P_2 \rangle_\chi$	=	$\langle P_1 \rangle_\chi \wedge \neg \langle P_2 \rangle_\chi$

The μ-calculus formulas with graded modalities provide a natural way to extend XPath expressions with counting constraints on children nodes, as exemplified by the following expression and its corresponding logical representation.

$$\mathbf{ancestor::p_1}[\mathit{count}(\text{child} :: p_2) \leq 5]$$
$$(\mathbf{p_1} \wedge \langle \mathbf{1} \rangle \mu \mathbf{x}.\top \vee \langle \mathbf{1} \rangle \mathbf{x} \vee \langle \mathbf{2} \rangle \mathbf{x}) \wedge p_2^{\leq 5}$$

Full translation of Core XPath with counting constraints on children is given in Figure 3.18. Auxiliary definitions are given in Figure 3.19, Figure 3.20 and Figure 3.17.

Notice that counting expressions like:

$$\mathit{count}(\text{child} :: p[Q]) \# k$$

are supported with no restriction on the qualifier Q, for instance, nested counting expressions are supported.

Lemma 3.5.1. *XPath expressions with counting constraints on children are linearly translated into enriched μ-calculus.*

Proof. We proceed by induction on the structure of the XPath expression. [Genevès et al., 2007] showed all the cases except when the qualifier is a counting expression. Hence, we address these additional cases.

The case when the qualifier is $\mathit{count}(\text{child} :: p) \# k$ is trivial. For the case $\mathit{count}(\text{child} :: p[Q]) \# k$, we know by induction that Q is linearly translated, then the full expression is also linearly translated. □

Figure 3.19 Logic embedding of Qualifiers

$$
\begin{array}{lcl}
\langle \text{Axis} \rangle^q_\chi & = & \left\langle \overline{\text{Axis}} \right\rangle_\chi \\
\langle \text{Axis} :: \text{QName} \rangle^q_\chi & = & \langle \text{Axis} \rangle^q_{\chi \wedge \langle \text{QName} \rangle} \\
\langle P_1 / P_2 \rangle^q_\chi & = & \langle P_1 \rangle^q_{\langle P_2 \rangle^q_\chi} \\
\langle P[Q] \rangle^q_\chi & = & \langle P \rangle^q_{\chi \wedge \langle Q \rangle^q_\top} \\
\langle Q_1 \text{ and } Q_2 \rangle^q_\chi & = & \langle Q_1 \rangle^q_\chi \wedge \langle Q_2 \rangle^q_\chi \\
\langle Q_1 \text{ or } Q_2 \rangle^q_\chi & = & \langle Q_1 \rangle^q_\chi \vee \langle Q_2 \rangle^q_\chi \\
\langle \text{not } Q \rangle^q_\chi & = & \neg \langle Q \rangle^q_\chi \\
\langle \mathit{count}(\text{child} :: \text{QName}) \# k \rangle^q_\chi & = & (\langle \text{QName} \rangle)^{\# k} \\
\langle \mathit{count}(\text{child} :: \text{QName}[Q]) \# k \rangle^q_\chi & = & (\langle \text{QName} \rangle \wedge \langle Q \rangle^q_\top)^{\# k}
\end{array}
$$

Figure 3.20 Dual Axis in XPath

$\overline{\overline{\text{Axis}}}$	=	Axis	$\overline{\text{self}}$	=	self
$\overline{\text{child}}$	=	parent	$\overline{\text{following-sibling}}$	=	preceding-sibling
$\overline{\text{ancestor}}$	=	descendant	$\overline{\text{ancestor-or-self}}$	=	descendant-or-self
$\overline{\text{following}}$	=	preceding			

3.6 Application to Regular Types

A tree schema denotes a set of trees satisfying certain structural constraints. In the XML context, several schema languages are actively used by applications:

- DTDs [Bray et al., 2004],
- XML Schema (W3C) [Fallside and Walmsley, 2004], and
- RELAX NG [Clark and Murata, 2001].

Regular tree types are a tree schema formalism that captures all XML schema languages used in practice [Murata et al., 2005].

Numerical restrictions on the number of children nodes can also be expressed by regular tree types. Actually, XML Schema provides explicit notations for such constraints using the following attributes: MinOccur and MaxOccur. This kind of counting constraints does not go beyond the expressivity

Figure 3.21 Syntax of Regular Tree Types

$$\begin{array}{lll} \text{TypeExpr} & := & \epsilon \mid \text{TypeExpr}_1 | \text{TypeExpr}_2 \mid p(x_1, x_2) \\ & & p(x_1^{\#k}, x_2) \mid \text{let } \overline{x}.\overline{\text{TypeExpr}} \text{ in TypeExpr} \end{array}$$

of regular languages, however, they provide a succinct notation for otherwise exponentially larger expressions.

In this Section, we first formally define regular types with counting constraints, and then we show those types can be embedded into μ-calculus with graded modalities.

We later define some common decision problems in the XML setting that involve the analysis of XPath expressions and XML schemas. Then, via the translation of XPath expressions and regular types into enriched μ-calculus, we show how XML static analysis problems, with counting constraints, can be decided in exponential time.

3.6.1 Type Expressions

Analogously to the isomorphism between n-ary and binary trees (Figure 3.2), there is also a straightforward isomorphism between regular types for n-ary trees and regular types for binary trees, as presented by [Genevès, 2006]. Hence, without loss of generality we consider types for binary trees as presented by the syntax described in Figure 3.21.

Regular types are interpreted as sets of tree structures. Their formal semantics is given in Figure 3.22, where V is a valuation function from variables to sets of trees, and $\text{lfp}(f)$ is the least fixpoint of f defined as follows: $f(V') = V\left[\overline{x}/\overline{[\![\mathcal{T}]\!]_{V'}}\right]$. Note that function f is monotone according to subset ordering, hence by the fixpoint theorem ([Tarski, 1955]), f has always a fixpoint.

Other traditional operators, like the Kleene star, are just syntactic sugar. For instance, if we want to express the recursive concatenation of a label p, we write the following expression.

$$\text{let } x_1, x_2.p(x_2, x_1)|\epsilon, \epsilon \text{ in } x_1$$

Recall also that the counting operator does not add expressivity to regular types, as illustrated by the following equivalent expressions.

$$\text{let } x_1, x_2, x_3.p_1(x_2^{=2}, x_3), p_2(x_3, x_3), \epsilon \text{ in } x_1$$
$$\text{let } x_1, x_2, x_3, x_4.p_1(x_2, x_4), p_2(x_4, x_3), p_2(x_4, x_4), \epsilon \text{ in } x_1$$

The advantage of the counting operator is that it produces exponentially smaller expressions.

Figure 3.22 Semantics of Regular Tree Types

$$
\begin{array}{lcl}
[\![\epsilon]\!]_V & = & \{\varnothing\} \\
[\![p(x_1, x_2)]\!]_V & = & \{K \mid \text{ the root of } K \text{ is labelled by } p, \\
 & & \text{its first children is in } V(x_1), \text{ and} \\
 & & \text{its following sibling is in } V(x_2)\} \\
[\![\mathcal{T}_1|\mathcal{T}_2]\!]_V & = & [\![\mathcal{T}_1]\!]_V \cup [\![\mathcal{T}_2]\!]_V \\
\left[\!\left[p(x_1^{\#k}, x_2)\right]\!\right]_V & = & \{K \mid \text{ the root of } K \text{ is labelled by } p, \\
 & & \text{the amount of its children in } V(x_1) \text{ satisfies } \#k, \text{ and} \\
 & & \text{its following sibling is in } V(x_2)\} \\
\left[\!\left[\text{let } \overline{x}.\overline{\mathcal{T}} \text{ in } \mathcal{T}\right]\!\right]_V & = & [\![\mathcal{T}]\!]_{\text{lfp}(f)}
\end{array}
$$

3.6.2 Types Embedding

We will show now that regular tree types with counting operators can be translated into μ-calculus with graded modalities. This translation is linear with respect to the size of the original type expression.

The relations involved in tree types are first child and following sibling. These relations are captured by modalities 1 and 2. Recursion in types is captured by the least fixpoint. We illustrate this correspondence between μ-calculus and types with the following example.

$$\text{let } x_1, x_2, x_3.p_1(x_2^{>k}, x_3), p_2(x_3, x_3), \epsilon \text{ in } x_1$$

This expression denotes the trees where p_1 nodes, with no siblings, have at least $k+1$ children nodes labelled wih p_2. We translate the expression into the logic as follows.

$$p_1 \wedge \neg\langle 2\rangle\top \wedge (p_2 \wedge \neg\langle 1\rangle\top)^{>k}$$

A translation function for arbitrary types is given in Figure 3.23. Auxiliary definitions required for the translation can be found in Figure 3.24.

Lemma 3.6.1. *For a given regular tree type expression, the lean size of its translation into enriched μ-calculus is linear with respect to the original expression size.*

Proof. We proceed by induction on the structure of the type expression.

For all the cases but the counting one, [Genevès et al., 2007] have already shown that the translation size is linear with respect to the size of the original type expression. We then consider the counting case.

Since variables must occur bounded, then let variables x_1 and x_2 be bounded by $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, in the type $p(x_1^{\#k}, x_2)$. We know by induction that

Figure 3.23 Embedding of Regular Tree Types into Enriched μ-calculus

$$
\begin{array}{lcl}
\langle \epsilon \rangle & = & \neg\top \\
\langle \mathcal{T}_1 | \mathcal{T}_2 \rangle & = & \langle \mathcal{T}_1 \rangle \vee \langle \mathcal{T}_2 \rangle \\
\langle p(x_1, x_2) \rangle & = & p \wedge succ_1(x_1) \wedge succ_2(x_2) \\
\left\langle p(x_1^{\#k}, x_2) \right\rangle & = & p \wedge x_1^{\#k} \wedge succ_2(x_2) \\
\langle \text{let } \overline{x}.\overline{\mathcal{T}} \text{ in } \mathcal{T} \rangle & = & \mu\overline{x}.\overline{\langle \mathcal{T} \rangle^{?}} \text{ in } \langle \mathcal{T} \rangle \\
\langle \mathcal{T} \rangle^{?} & = & \begin{cases} \langle \mathcal{T} \rangle^{\#} & \text{if } x, \text{ bounded to } \mathcal{T}, \\ & \text{ is under the scope of a } \# \text{ operator} \\ \langle \mathcal{T} \rangle & \text{otw.} \end{cases} \\
\langle \epsilon \rangle^{\#} & = & \neg\top \\
\langle \mathcal{T}_1 | \mathcal{T}_2 \rangle^{\#} & = & \langle \mathcal{T}_1 \rangle^{\#} \vee \langle \mathcal{T}_2 \rangle^{\#} \\
\langle p(x_1, x_2) \rangle^{\#} & = & p \wedge succ_1(x_1) \wedge succ^{\#}(x_2) \\
\left\langle p(x_1^{\#k}, x_2) \right\rangle^{\#} & = & p \wedge x_1^{\#k} \wedge succ^{\#}(x_2) \\
\langle \text{let } \overline{x}.\overline{\mathcal{T}} \text{ in } \mathcal{T} \rangle^{\#} & = & \mu\overline{x}.\overline{\langle \mathcal{T} \rangle^{\#}} \text{ in } \langle \mathcal{T} \rangle^{\#}
\end{array}
$$

the leans of the following formulas are linear w.r.t. their corresponding subexpressions.

$$p \qquad\qquad \langle \mathcal{T}_1 \rangle^{\#} \qquad\qquad succ_2(x_2)$$

Recall that the lean of $(\langle \mathcal{T}_1 \rangle^{\#})^{\#k}$ is the same as the one of $\langle 1 \rangle \mu x. \langle \mathcal{T}_1 \rangle^{\#} \vee \langle 2 \rangle x$, hence only two additional modal formulas are introduced in the lean.

□

3.6.3 XML decision problems

With the previous translations, we are now able to translate both XPath and Regular type expressions into μ-calculus. Several decision problems for the involving static analysis of XPath and XML schemas can be solved via this translation.

We need a special proposition Ⓢ in the translation of XPath expressions. This proposition marks the initial context node from which an XPath expression is evaluated. Since this context is unique in the tree, it may serves to compare several XPath expressions, e.g. for testing the equivalence of expressions.

Figure 3.24 Auxiliary Functions in the Translation of Regular Tree Types

$$
\begin{array}{lcl}
succ_m & = & \begin{cases} \neg\langle m\rangle\top & \text{if } nullable(x) \\ \langle m\rangle x & \text{if not } nullable(x) \end{cases} \\
succ^{\#} & = & \begin{cases} \top & \text{if } nullable(x) \\ \langle 2\rangle x & \text{if not } nullable(x) \end{cases} \\
nullable(x) & = & nullable(\mathcal{T}) \qquad x \text{ is bound to } \mathcal{T} \\
nullable(\epsilon) & = & \text{true} \\
nullable(\mathcal{T}_1|\mathcal{T}_2) & = & nullable(\mathcal{T}_1) \text{ and } nullable(\mathcal{T}_2) \\
nullable(p(x_1, x_2)) & = & \text{false} \\
nullable(p(x_1^{\#k}, x_2)) & = & \text{false} \\
nullable(\text{let } \overline{x}.\overline{\mathcal{T}} \text{ in } \mathcal{T}) & = & nullable(\mathcal{T})
\end{array}
$$

XML decision problems are now defined in terms of the μ-calculus. In this context, an expression can be either: an XPath expression, a regular type expression, or an XPath expression under some regular type expression.

- Containment. An expression is said to be contained into another expression when the first expression always selects nodes that are selected by the second expression. We say the XPath expression P_1 is contained in the XPath expression P_2, written $P_1 \subseteq P_2$, when the formula $\langle P_1\rangle \wedge \neg\langle P_2\rangle$ is unsatisfiable. Moreover, we can test XPath containment under schema constraints (type expression): $P_1(\mathcal{T}_1) \subseteq P_2(\mathcal{T}_2)$ iff $\langle \mathcal{T}_1\rangle \wedge \langle P_1\rangle \wedge \neg(\langle \mathcal{T}_2\rangle \wedge \langle P_2\rangle)$ is not satisfiable.

- Equivalence. Two expressions are equivalent if one is contained into the other, and vice versa.

- Emptiness. An expression is empty when it always selects the empty set. For testing non emptiness, we write $P \neq \varnothing$, $\mathcal{T} \neq \varnothing$, and $P(\mathcal{T}) \neq \varnothing$, respectively, whenever the respective formula $\langle P\rangle$, $\langle \mathcal{T}\rangle$, or $\langle P(\mathcal{T})\rangle$ is not satisfiable.

- Overlap. Two expressions overlap if they both select a common node. This can be tested by checking the unsatisfiability of the conjunction of the translations of both expressions.

- Coverage. Given a finite sequence of expressions $e_1, \ldots, e_k$, e_1 is covered by $e_2, \ldots, e_k$ iff e_1 is contained in the union of e_i $(i = 2, \ldots, k)$. Recall e_1 can be an XPath expression, a type or an XPath expression under some schema.

Theorem 3.6.1. *In the context of XPath and XML schema languages with cardinality constraints on children nodes, the XML decision problems of containment, equivalence, emptiness, overlap and coverage are decidable in exponential time with respect to the problem size.*

Proof. Immediate from Theorem 3.4.4, Lemma 3.5.1 and Lemma 3.6.1. □

3.7 Conclusion

The logic introduced in this Chapter is the alternate-free μ-calculus for trees, with converse, nominals and graded modalities. Although the logic does not go beyond the expressive power of simple μ-calculus, it provides notations with a doubly exponential gain in succinctness.

The main contributions of this Chapter are twofold: the logic is shown to be decidable, and XPath and XML schema translations into the logic are provided. Decidability of the logic is shown via a tableau-based algorithm with a single exponential complexity. The characterization of the XML languages provides an efficient framework for expressions with counting constraints on children nodes.

One may now wonder whether it is possible to express counting constraints beyond children nodes, for instance, descendants. In [Bianco et al., 2009], it was shown that imposing constraints, via μ-calculus with graded modalities, on descendant nodes comes at exponential costs for succinctness. This exponential blow-up implies that satisfiability of expressions with counting constraints on descendants is in double exponential time.

In Chapter 4, we equip μ-calculus with counting operators able to express constraints on nodes which are reachable by means of regular paths, for instance, descendant or ancestor nodes.

Chapter 4

μ-calculus with graded paths

In this Chapter, we extend μ-calculus with counting operators which are more general than those of graded modalities, which are limite in scope to children nodes. In the logic proposed here, it is possible to constrain the number of multi-directional recursive paths. For instance, the formula $\langle 1\rangle\langle(1|2)^\star\rangle\phi^{>10}$ holds at nodes with more than 10 descendants where ϕ holds.

It is has been already observed by [Bianco et al., 2009] that this kind of counting constraints can be exponentially more succinct than graded modalities.

Previous work of [Bonatti et al., 2006] shows that extending μ-calculus with graded modalities and nominals over general graph models is undecidable. We showed in Chaper 3 that such result does not apply when we consider tree models. Here, we extended decidability of μ-calculus with more general counting constraints. In particular, we present a tableau-based algorithm for the satisfiability of the logic with an exponential complexity. The logic here can perform a more general kind of counting than graded modalities of Chapter 3, but we restrict the nesting of counting formulas in other formulas. This restriction make it difficult to compare the logic here with the one of Chapter 3.

We present in this chapter also a linear embedding of XPath expressions in the presence of XML structural constraints. XPath and XML Type expressions, in contrast with the ones defined in Chapter 3, are able to express counting constraints on arbitrary regular paths.

4.1 The Tree Logic

As in the previous chapter, we extended the alternation-free μ-calculus, originally introduced by [Genevès et al., 2007], with counting constraints. The main features of the logic are the following:

Figure 4.1 Syntax of Trails

$$\begin{array}{lllllll} \alpha & := & \alpha_0 & | & \alpha_0^\star & | & \alpha_0^\star\alpha \\ \alpha_0 & := & m & | & \alpha_0\alpha_0 & | & \alpha_0|\alpha_0 \end{array}$$

- a fixpoint operator, that allows to perform recursive navigation;
- converse modalities, which allow to perform backward navigation;
- counting formulas, which allow to efficiently express counting constraints on non contiguous nodes; and
- nominals, which denote singleton subsets of the tree model.

Before defining the logic, we first introduce the notion of trails. Trails will serve to denote the navigation performed by counting formulas.

4.1.1 Trails

Trails are defined as regular expressions formed by modalities only (see Figure 4.1).

For the form of trails, instead of considering the traditional regular expression form, we define a *safe* form in Figure 4.1. This form will be used to show that our satisfiability algorithm for the logic is complete. The notion of cycle-free trails is also considered. In order to properly define such notion, we first extract the set of sequences of modalities (paths) produced by a trail with the following function.

$$\begin{array}{llllll} paths(m) & = & \{m\} & paths(\alpha_1\alpha_2) & = & \{\rho_1\rho_2 \mid \rho_i \in \alpha_i\} \\ paths(\alpha_1 \mid \alpha_2) & = & paths(\alpha_1) \cup paths(\alpha_2) & paths(\alpha^\star) & = & paths(\alpha\alpha^\star) \end{array}$$

Definition 4.1.1 (Cycle-free trails)**.** We say a trail α is *cycle-free*, when for every sequence $\rho \in paths(\alpha)$, we have that the set $cycles(\rho)$ is finite.

We will use trails to represent multi-directional recursive navigation over trees. More precisely, trails denote sets of paths among two nodes in tree structures.

Definition 4.1.2. In a given tree K, we say there is a trail α from a node n_1 to the node n_2, written $n_1 \xrightarrow{\alpha} n_2$, if and only if there is a sequence of nodes $n_0, \ldots, n_k$ and a path $\rho = m_1 \ldots m_k$, such that $\rho \in paths(\alpha)$, and $\mathcal{R}^K(n_j, m_{j+1}) = n_{j+1}$ for every $j = 0, \ldots, k-1$.

Figure 4.2 Graded Paths Logic Syntax

$$
\begin{array}{lllllllllll}
\Phi \ni \phi & := & p & | & \top & | & x & | & \neg p & | \\
 & & \neg\top & | & \neg\langle m\rangle\top & | & \phi \wedge \phi & | & \phi \vee \phi & | \\
 & & \langle m\rangle\phi & | & \mu\overline{x}.\overline{\psi} \text{ in } \psi & | & \langle\alpha\rangle\psi^{\#k} & & & \\
\psi & := & p & | & \top & | & x & | & \neg p & | \\
 & & \neg\top & | & \neg\langle m\rangle\top & | & \psi \wedge \psi & | & \psi \vee \psi & | \\
 & & \langle m\rangle\psi & | & \mu\overline{x}.\overline{\psi} \text{ in } \psi & & & & &
\end{array}
$$

Figure 4.3 Negated Normal Form of Graded Paths Formulas

$$
\begin{array}{lcl}
\neg\langle m\rangle\phi & \equiv & \neg\langle m\rangle\top \vee \langle m\rangle\neg\phi \\
\neg\mu\overline{x}.\overline{\phi} \text{ in } \phi & \equiv & \mu\overline{x}.\overline{\neg\phi\left[{}^{x}/_{\neg x}\right]} \text{ in } \phi\left[{}^{\overline{x}}/_{\neg\overline{x}}\right] \\
\neg\langle\alpha\rangle\phi^{>k} & \equiv & \langle\alpha\rangle\phi^{\leq k} \\
\neg\langle\alpha\rangle\phi^{\leq k} & \equiv & \langle\alpha\rangle\phi^{>k}
\end{array}
$$

4.1.2 Syntax and Semantics

The syntax of the logic is given in Figure 4.2, where m is a modality, k is a natural number, and $\# \in \{\leq, >\}$. A formula written ϕ may contain counting subformulas, whereas formulas written ψ cannot. We thus disallow counting under counting or under fixpoints.

The syntax of this logic follows the same intuition than the one defined in Figure 3.3. A fixed signature of propositions, modalities and variables is considered. In contrast to the logic of Figure 3.3, formulas of the form $\phi^{\#k}$ are now of the form $\langle\alpha\rangle\psi^{\#k}$.

The syntax in Figure 4.2 is also given in negation normal form defined by the usual DeMorgan's rules and the rules of Figure 4.3.

We also consider here only cycle-free formulas. In order to define cycle-free formulas we first need to update our notion of the trail of a formula.

Definition 4.1.3 (Trail of a formula)**.** The *trail of a formula* is the set of sequences of modalities generated by the unfolding of the fixpoints, and it is

Figure 4.4 Syntactic sugar for Graded Paths Formulas

$\neg\phi$	$\equiv$	negated normal form of ϕ
$\mu x.\phi$	$\equiv$	$\mu x.\phi$ in x
$\langle\alpha\rangle\phi^{=k}$	$\equiv$	$\langle\alpha\rangle\phi^{\leq k} \wedge \langle\alpha\rangle\phi^{>k-1}$

defined as follow:

$$trail(p) = trail(\neg p) = trail(\top) = trail(\neg\top) = trail(\neg\langle m\rangle\top) = \epsilon$$
$$trail(\langle m\rangle\phi) = \{m \cdot \rho \mid \rho \in trail(\phi)\}$$
$$trail(\langle\alpha\rangle\phi^{\#k}) = \{\rho_1 \cdot \rho_2 \mid \rho_1 \in paths(\alpha), \rho_2 \in trail(\phi)\}$$
$$trail(\phi_1 \wedge \phi_2) = trail(\phi_1 \vee \phi_2) = trail(\phi_1) \cup trail(\phi_2)$$
$$trail(\mu x.\phi) = trail(\phi\left[{}^{\mu x.\phi}/_{x}\right])$$

Definition 4.1.4 (Cycle-free formulas)**.** We say a formula ϕ is cycle-free, when for every $\rho \in trail(\phi)$, the set $cycles(\rho)$ is finite.

We also consider the syntactic sugar from Figure 4.4.

Formulas are interpreted as subsets of nodes in a tree structure K. A counting formula $\langle\alpha\rangle\phi^{>k}$ satisfied at a node n means that there are at least $k+1$ different nodes satisfying ϕ that can be reached from n through the trail α. Hence the formula $\langle\alpha\rangle\phi^{>k}$ is interpreted as the as the set of nodes such that, for each of them, the previously described condition holds. Formal semantics of the logic is depicted in Figure 4.5.

In order to illustrate the intepretation of formulas, consider the following formula.

$$p_1 \wedge \langle 1\rangle\langle(1|2)^\star\rangle p_2^{\geq 4}$$

This formula holds at nodes labeled by p_1 that have more than 4 descendant nodes labeled by p_2. A graphical representation of this example is depicted in Figure 4.6.

4.1.3 Encoding of graded paths

For the satisfiability algorithm in Section 4.2, we need to show there is a finite unfolding for every fixpoint formula. For that purpose, we first encode counting formulas into plain μ-calculus (with no counting operator), then we use the fact that there is a finite unfolding for every fixpoint formula in the plain μ-calculus.

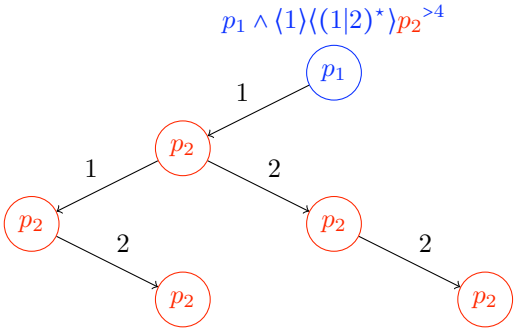
First notice that the navigation performed by trails can also be done by plain μ-calculus formulas.

$$\langle 1^\star\rangle\phi^{>0} \equiv \mu x.\phi \vee \langle 1\rangle x$$

Figure 4.5 Graded Paths Logic Semantics

$$
\begin{array}{lcl}
[\![p]\!]^K_V & = & \{n \mid \mathcal{L}^K(n) = p\} \\
[\![\neg p]\!]^K_V & = & \{n \mid \mathcal{L}^K(n) \neq p\} \\
[\![\top]\!]^K_V & = & \mathcal{N}^K \\
[\![\neg\top]\!]^K_V & = & \varnothing \\
[\![\neg\langle m\rangle\top]\!]^K_V & = & \{n \mid \mathcal{R}^K(n,m) \text{ is not defined}\} \\
[\![\phi_1 \wedge \phi_2]\!]^K_V & = & [\![\phi_1]\!]^K_V \cap [\![\phi_2]\!]^K_V \\
[\![\phi_1 \vee \phi_2]\!]^K_V & = & [\![\phi_1]\!]^K_V \cup [\![\phi_2]\!]^K_V \\
[\![\langle m\rangle\phi]\!]^K_V & = & \{n \mid \mathcal{R}^K(n,m) \in [\![\phi]\!]^K_V\} \\
[\![\mu\overline{x}.\overline{\psi} \text{ in } \psi]\!]^K_V & = & [\![\psi]\!]^K_{V[\overline{\mathcal{N}''}/\overline{x}]}, \text{where } \overline{\mathcal{N}''} = \bigcap\{\overline{\mathcal{N}'} \mid \overline{[\![\psi]\!]^K_{V[\overline{\mathcal{N}'}/\overline{x}]}} \subseteq \overline{\mathcal{N}'}\} \\
[\![\langle\alpha\rangle\psi^{\#k}]\!]^K_V & = & \{n \mid |n' \in [\![\psi]\!]^K_V \mid n \xrightarrow{\alpha} n'|\#k\}
\end{array}
$$

Figure 4.6 Intepretation of a graded paths formula



Navigation performed by trails is defined by the nav function of Figure 4.7.

In order to count different nodes satisfying the same formula, we now make use of an order, called document ordering, as already presented by [Afanasiev et al., 2005].

Definition 4.1.5. We define the next function as follows:

$$\text{next}^1(\phi) \equiv \text{next}(\phi) \equiv \text{nav}(11^\star|(\overline{1}^\star 22^\star 1^\star), \phi)$$
$$\text{next}^{k+1}(\phi) \equiv \text{next}^k(\phi \wedge \text{next}(\phi))$$

The navigation performed by the trail $11^\star|(\overline{1}^\star 22^\star 1^\star$ is a left-depth first

Figure 4.7 Extracting the navigational information of trails

$$\mathrm{nav}(\phi) = \phi \qquad \text{for } \phi = p, \neg p, x, \top, \neg\top, \neg\langle m\rangle\top$$
$$\mathrm{nav}(\phi_1 \wedge \phi_2) = \mathrm{nav}(\phi_1) \wedge \mathrm{nav}(\phi_2) \qquad \mathrm{nav}(\phi_1 \vee \phi_2) = \mathrm{nav}(\phi_1) \vee \mathrm{nav}(\phi_2)$$
$$\mathrm{nav}(\langle m\rangle\phi) = \langle m\rangle\mathrm{nav}(\phi) \qquad \mathrm{nav}(\mu x.\phi) = \mu x.\mathrm{nav}(\phi)$$
$$\mathrm{nav}(\langle\alpha\rangle\phi^{\#k}) = \mathrm{nav}(\alpha, \phi)$$

$$\begin{aligned}
\mathrm{nav}(m, \phi) &= \langle m\rangle\mathrm{nav}(\phi)\\
\mathrm{nav}(\alpha_1\alpha_2, \phi) &= \mathrm{nav}(\alpha_1, \mathrm{nav}(\alpha_2, \mathrm{nav}(\phi)))\\
\mathrm{nav}(\alpha_1|\alpha_2, \phi) &= \mathrm{nav}(\alpha_1, \mathrm{dc}(\phi)) \vee \mathrm{nav}(\alpha_2, \mathrm{nav}(\phi))\\
\mathrm{nav}(\alpha^\star, \phi) &= \mu x.\mathrm{nav}(\phi) \vee \mathrm{nav}(\alpha, x)
\end{aligned}$$

traversal of the tree. Note now that the formula $\mathrm{next}^k(\phi)$ holds at the root of tree models such that there are exactly k different nodes where ϕ holds.

We now get rid of counting operators.

Definition 4.1.6. We define the *discount* transformation function as follows:

$$\begin{aligned}
\mathrm{dc}(\langle\alpha\rangle\psi^{>k}) &= @\mathrm{n} \wedge \mathrm{nav}(\alpha, \psi) \wedge \mathrm{nav}((\overline{1}|\overline{2})^\star, \mathit{root} \wedge \mathrm{next}^{k+1}(\psi \wedge \mathrm{nav}(\overline{\alpha}, @\mathrm{n}))\\
\mathrm{dc}(\langle\alpha\rangle\psi^{\leq k}) &= \neg\mathrm{dc}(\langle\alpha\rangle\psi^{>k})
\end{aligned}$$

where @n is a fresh nominal introduced by each counting subformula, the trail $(1|2)^\star$ goes to the root node, which is defined by the formula $\mathit{root} = \neg\langle\overline{1}\rangle\top \wedge \neg\langle 2\rangle\top$, and $\overline{\alpha}$ is the inverse trail as defined as follows.

$$\overline{\alpha_1\alpha_2} = \overline{\alpha_2\alpha_1} \qquad \overline{\alpha_1|\alpha_2} = \overline{\alpha_1}|\overline{\alpha_2} \qquad \overline{\alpha^\star} = \overline{\alpha}^\star$$

We now show that the coding introduced by the discount function comes at exponential cost. An updated notion of formula size is detailed in Figure 4.8.

Lemma 4.1.1. *A counting formula ϕ and its encoding $dc(\phi)$ are equivalent, and the size of $dc(\phi)$ is exponentially larger than the size of ϕ.*

Proof sketch. The equivalence is proven by a double induction: on the structure, and on the numerical constraint of the counting formula.

Regarding the exponential blow-up, note that size of $\mathrm{next}^k(\psi)$ directly depends on the numerical constraint k. If the number k is encoded in binary, then the size $\mathrm{next}^k(\psi)$ increases exponentially w.r.t. k. □

We now can conclude a double exponential satisfiability for the logic.

Theorem 4.1.1. *The logic is satisfiable in double exponential time.*

Figure 4.8 Formula size in μ-calculus with graded paths

$$
\begin{aligned}
&|p| = |\top| = |x| = |m| = 1\\
&|\neg\phi| = |\mu x.\phi| = |\phi|\\
&|\langle m\rangle\phi| = |\phi| + 1\\
&|\phi_1 \wedge \phi_2| = |\phi_1 \vee \phi_2| = |\phi_1| + |\phi_2|\\
&|\langle\alpha\rangle\phi^{\#k}| = |\phi| + k + 1 + |\alpha|\\
&|\alpha_1\alpha_2| = |(\alpha_1|\alpha_2)| = |\alpha_1| + |\alpha_2|\\
&|\alpha^\star| = |\alpha|
\end{aligned}
$$

4.1.4 Fixpoint unfolding and normal forms

With a formal notion of size, we can now state that the negated normal form is linear with respect to the original formula.

Lemma 4.1.2 (Negation Normal Form)**.** *A formula and its negation normal form are equivalent, furthermore, the negation normal form has a linear size with respect to the original formula.*

Since we now know the logic is satisfiable (Theorem 4.1.1), and the logic without counting operators has a finite unfolding (Definition 3.2.4), for every fixpoint formula ([Genevès et al., 2007]), we can now state a finite unfolding property for our logic.

Lemma 4.1.3. *There is a finite unfolding for every fixpoint formula.*

4.1.5 Global counting formulas

To conclude this section, we turn to an illustration of the expressive power of the logic. An interesting consequence of the inclusion of backward axes in trails is the ability to reach every node in the tree from any node of the tree, using the trail $(\overline{1}|\overline{2})^\star(1|2)^\star$. We can thus select some nodes based on some global counting property. Consider the following formula, where # stands for one of the comparison operators $\leq, >$ or $=$.

$$\langle(\overline{1}|\overline{2})^\star(1|2)^\star\rangle\phi_1^{\#k}$$

Intuitively, this formula counts how many nodes in the whole tree satisfy ϕ_1. For each node of the tree, it selects it if and only if the count is compatible with the comparison considered. The interpretation of this formula is thus either every node of the tree, or none. It is then easy to restrict the selected nodes to some other formula ϕ_2, using intersection.

$$\langle(\overline{1}|\overline{2})^\star(1|2)^\star\rangle\phi_1^{\#k} \wedge \phi_2$$

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

$$\phi_1 \# k \Longrightarrow \phi_2$$

We can now express existential properties, such as: select every node satisfying ϕ_2 if there exists a node satisfying ϕ_1.

$$\phi_1 > 0 \Longrightarrow \phi_2$$

Universal properties can also be expressed: select every node satisfying ϕ_2 if every node satisfies ϕ_1.

$$\neg\phi_1 \leq 0 \Longrightarrow \phi_2$$

An alternative definition of nominals, to the one in Subsection 3.2.6, is by means of the following counting formulas:

$$@\mathrm{n} \equiv (n > 0 \Longrightarrow n)$$

4.2 Satisfiability Algorithm

We present here the corresponding algorithm for checking satisfiability of formulas. Given a formula, the algorithm seeks to build a tree containing a node selected by the formula. We show that our algorithm is correct and complete: a satisfying tree is found if and only if the formula is satisfiable. We also show that the time complexity of the algorithm is exponential in the size of the formula.

4.2.1 Overview

Similarly to the algorithm introduced at Section 3.3, the algorithm defined in this section 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. More precisely, the lean mainly gathers atomic propositions and modal subformulas. From the lean, one may gather a finite number of formulas, called a ϕ-node, which may be satisfied at a given node of a tree. Trees of ϕ-nodes represent the exhaustive search universe in which the algorithm searches for a satisfying tree.

The second stage of the algorithm consists in the building of sets of such trees in a bottom-up manner, ensuring consistency at each step. Initially, all possible leaves (i.e., ϕ-node that do not have children nodes) are considered. During further steps, the algorithm considers every possible ϕ-node that can be connected with a tree of the previous steps, while maintaining consistency. For instance, if a formula at a ϕ-node n involves a forward modality $\langle 1\rangle\phi'$, then ϕ' must be verified at the first child of n. Reciprocally, due to converse modalities,

a ϕ-node may impose restrictions on its possible parent nodes. The new trees that are built may involve converse modalities also, which will be satisfied during further steps of the algorithm. To ensure the algorithm terminates, a bound on the number of times each ϕ-nodee may occur in the tree is given.

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
- every tree has been considered (the exploration of the whole search universe is complete): the formula is unsatisfiable.

The main difference with respect to the algorithm of Section 3.3 is that the machinery in charge of handling counting formulas is generalized in order to support general trails instead of only children-trails.

4.2.2 Preeliminaries

Two notions are defined here: ϕtrees, and an entailment relation among ϕtrees and formulas. ϕtrees are intended to be syntactic versions of tree models. In order to define ϕtrees, we also need to define syntactic versions of nodes and the transition relation of between nodes. All information needed to build the ϕtrees is extracted from formulas, hence we first define the machinery required to extract such information.

Definition 4.2.1 (Fischer-Ladner Closure)**.** For a given formula ϕ, the *Fischer-Ladner closure* $\mathrm{FL}(\phi)$ is defined as the following set.

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

where k is the smallest integer such that $\mathrm{FL}(\phi)_{k+1} = \mathrm{FL}(\phi)_k$, and R^{fl} is defined as follow:

$$\begin{array}{ll} R^{\mathrm{fl}}(\phi_1 \wedge \phi_2, \phi_i) & R^{\mathrm{fl}}(\phi_1 \vee \phi_2, \phi_i) \\ R^{\mathrm{fl}}(\langle m\rangle\psi, \psi) & R^{\mathrm{fl}}\left(\mu x.\psi, \psi\left[{}^{\mu x.\psi}/_{x}\right]\right) \\ R^{\mathrm{fl}}(\langle\alpha\rangle\psi^{>k}, \mathrm{nav}(\psi \wedge c)) & R^{\mathrm{fl}}(\langle\alpha\rangle\psi^{\leq k}, \mathrm{nav}((\psi \wedge c) \vee (\neg\psi \wedge \neg c)) \end{array}$$

where $i = 1, 2$, $\# \in \{>, \leq\}$, c is a fresh proposition, and nav is defined in Figure 4.7. The fresh proposition c will serve as an identifier for the nodes aimed to be counted for each counting subformula.

We now define the set containing the aimed propositional and navigational information composing the nodes.

Figure 4.9 Local entailment relation for graded paths

$$\frac{}{n \vdash \top} \qquad \frac{\phi \in n}{\Gamma(\rho) \vdash \phi} \qquad \frac{\phi \notin \Gamma(\rho)}{n \vdash \neg\phi}$$

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

$$\frac{n \vdash \phi\left[{}^{\mu x.\phi}/_{x}\right]}{n \vdash \mu x.\phi} \qquad \frac{n \vdash \mathrm{nav}(\alpha, \phi)}{n \vdash \langle\alpha\rangle\phi^{>k}} \qquad \frac{n \vdash \mathrm{nav}(\alpha, (\phi \wedge c) \vee (\neg\phi \wedge \neg c))}{n \vdash \langle\alpha\rangle\phi^{\leq k}}$$

Definition 4.2.2 (Lean set)**.** Consider a formula ϕ and a proposition σ not occurring in ϕ. We define the *lean* set of ϕ as follows.

$$lean(\phi) = \{p, \langle m\rangle\psi \in \mathrm{FL}(\phi)\} \cup \{\sigma, \langle m\rangle\top\}$$

The syntactic representation of nodes is now defined.

Definition 4.2.3 (ϕ-node)**.** A ϕ-node of a formula ϕ, written n^{ϕ}, is defined as a subset of $lean(\phi)$, such that:

- exactly one non nominal proposition is present;
- when $\langle m\rangle\psi$ occurs, also does $\langle m\rangle\top$; and
- both $\langle\overline{1}\rangle\top$ and $\langle\overline{2}\rangle\top$ cannot occur simultaneously.

The set of ϕ-nodes is denoted by N^{ϕ}. When the formula under consideration is fixed, we often omit the superscript.

With a clear definition of ϕ-nodes, we can now define the ϕtrees.

Definition 4.2.4 (ϕtree)**.** A ϕtree is either an empty tree $\varnothing$, or a triple $(n^{\phi}, \Gamma_1, \Gamma_2)$, where Γ_1 and Γ_2 are ϕtrees. When clear from the context, we refer to ϕtrees simply as trees.

We now turn to the definition of consistency of a ϕtree. To this end, we define in Figure 4.9 an entailment relation between a node and a formula.

The main purpose of the local entailment relation is to serve in the definition of the transition relation between two nodes.

Definition 4.2.5 (Modal consistency)**.** Given a formula ϕ, two ϕ-nodes n_1, n_2, we say n_1 and n_2 are *modally consistent*, written $\Delta(n_1, n_2)$, if and only if, for $i = 1, 2$, we have that

$$\forall\langle i\rangle\psi \in lean(\phi), \langle i\rangle\phi \in n_1 \iff n_2 \vdash \phi$$
$$\forall\langle\overline{i}\rangle\psi \in lean(\phi), \langle\overline{i}\rangle\phi \in n_2 \iff n_1 \vdash \phi$$

Figure 4.10 Global entailment relation for graded paths: trees and formulas

$$\frac{}{\Gamma(\rho) \Vdash \top} \qquad \frac{\phi \in \Gamma(\rho)}{\Gamma(\rho) \Vdash \phi} \qquad \frac{\phi \notin \Gamma(\rho)}{\Gamma(\rho) \Vdash \neg\phi} \qquad \frac{\Gamma(\rho) \Vdash \phi \quad \Gamma(\rho) \Vdash \psi}{\Gamma(\rho) \Vdash \phi \wedge \psi}$$

$$\frac{\Gamma(\rho) \Vdash \phi}{\Gamma(\rho) \Vdash \phi \vee \psi} \qquad \frac{\Gamma(\rho) \Vdash \psi}{\Gamma(\rho) \Vdash \phi \vee \psi} \qquad \frac{\Gamma(\rho) \Vdash \phi\left[{}^{\mu x.\phi}/_{x}\right]}{\Gamma(\rho) \Vdash \mu x.\phi}$$

$$\frac{|\{\Gamma(\rho\rho') \mid \rho' \in \alpha, \Gamma(\rho\rho') \Vdash \phi\}| > k}{\Gamma(\rho) \Vdash \langle\alpha\rangle\phi^{>k}}$$

$$\frac{|\{\Gamma(\rho\rho') \mid \rho' \in \alpha, \Gamma(\rho\rho') \Vdash \phi\}| \leq k \qquad \forall \rho' \in \alpha : \Gamma(\rho\rho') \Vdash (\phi \wedge c) \vee (\neg\phi \wedge \neg c)}{\Gamma(\rho) \Vdash \langle\alpha\rangle\phi^{\leq k}}$$

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 in the case of counting formulas, in the local entailment relation, it is only checked whether the nodes contains the modal information required to navigate to the counted nodes. Such information is represented by the nav function.

In order to properly check the satisfaction of counting formulas, we actually need to perform the navigation of the trails occurring in the counting formulas. For that purpose, we will refer to nodes in a non empty tree, by means of a unique path (sequences of modal symbols) leading to them from the root, in the following manner, where $i = 1, 2$.

$$(n, \Gamma_1, \Gamma_2)(\epsilon) = n \qquad (n, \Gamma_1, \Gamma_2)(\rho i) = \Gamma_i \qquad (n, \Gamma_1, \Gamma_2)(\rho i \bar{i}) = (n, \Gamma_1, \Gamma_2)$$

Now, we are able to define a global entailment relation as given in ϕtrees in Figure 4.10. This relation extends the local entailment relation with checks for counting formulas that actually count nodes.

We conclude the preliminaries by introducing some final notations. The root of a tree is defined as follows.

$$\begin{array}{lcl} root(\varnothing) & = & \varnothing \\ root((n, \Gamma_1, \Gamma_2)) & = & n \end{array}$$

A tree Γ *satisfies* a formula ϕ, if there is a path ρ, such that $\Gamma(\rho) \vdash \phi$, and neither $\langle\overline{1}\rangle\top$ nor $\langle\overline{2}\rangle\phi$ occur in $root(\Gamma)$. We write $\Gamma \Vdash \phi$, when Γ satisfies ϕ. Given a set of trees ST, we write $ST \Vdash \phi$, when there is a tree in ST satisfying ϕ.

Figure 4.11 Enriched μ-calculus: occurrences bound functions

$$maxK(p) = maxK(\neg p) = maxK(\neg\langle m\rangle\top) = maxK(\top) = maxK(\mu x.\phi) = 0$$
$$maxK(\phi_1 \wedge \phi_2) = maxK(\phi_1 \vee \phi_2) = maxK(\phi_1) + maxK(\phi_2)$$
$$maxK(\langle m\rangle\phi) = maxK(\phi)$$
$$maxK(\langle\alpha\rangle\phi^{\#k}) = k + 1$$

$$\begin{array}{lcl}
\mathrm{nmax}(n, \Gamma_1, \Gamma_2) & = & \max(\mathrm{nmax}(n, \Gamma_1), \mathrm{nmax}(n, \Gamma_2)) \\
\mathrm{nmax}(n, (n, \Gamma_1, \Gamma_2)) & = & 1 + \mathrm{nmax}(n, \Gamma_1, \Gamma_2) \\
\mathrm{nmax}(n, (n', \Gamma_1, \Gamma_2)) & = & \mathrm{nmax}(n, \Gamma_1, \Gamma_2), \text{where } n \neq n' \\
\mathrm{nmax}(n, \varnothing) & = & 0
\end{array}$$

4.2.3 The Algorithm

We are now ready to present the algorithm, which is parametrized by $maxK(\phi)$ (defined in Figure 4.11), the maximum number of occurrences of a given node in a path from the root of the trees to a leaf.

The algorithm builds consistent candidate trees 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 evetually stops and returns 0 if no satisfying tree has been found.

Algorithm 2 μ-Calculus with graded paths: Check Satisfiability of ϕ

```
ST ← ∅
repeat
  AUX ← {(n, Γ1, Γ2) |                      {we extend the trees}
       nmax(n, Γ1, Γ2) ≤ maxK(φ) + 2        {with an available node}
       Δ(n, root(Γi))                       {checking consistency}
       where Γi ∈ ST ∪ {∅} and i = 1, 2 }
  if AUX ⊆ ST then
    return 0                                {no new tree was built}
  end if
  ST ← ST ∪ AUX
until ST ⊩ φ
return 1
```

To bound the size of the trees that are built, we restrict the number of identical nodes in paths from the root to leaves by $maxK(\phi)$, defined in Figure 4.11. The function nmax is in charge of counting the identical nodes in paths from the root to the leaves, and it is also defined in Figure 4.11.

Recall that counting formulas tested by the algorithm of Subsection 3.3.3

counts only children nodes. This case is covered, in this Section, by the counting formulas with the trail $1, 2^\star$. Hence, since the algorithm introduced here operates basically in the same manner (triples are formed bottom up), the corresponding formula for the example of Figure 3.14 is $p_1 \wedge \langle 12^\star \rangle p_2^{\geq 2}$. The algorithm builds basically the same triples.

As another example, consider now the following formula.

$$p_1 \wedge \langle 1 \rangle \langle (1|2)^\star \rangle p_2^{\geq 4}$$

The lean is the following set, where $\psi = \mu x.(p_2 \wedge c) \vee \langle 1 \rangle x \vee \langle 2 \rangle x$, c is the counting proposition corresponding to $\langle (1|2)^\star \rangle p_2^{\geq 4}$, $m = 1, 2, \overline{1}, \overline{2}$, and p_3 is a proposition not occurring in the original formula.

$$\{p_1, p_2, c, p_3, \langle m \rangle \top, \langle 1 \rangle \langle (1|2)^\star \rangle p_2^{\geq 4}, \langle 1 \rangle \psi, \langle 2 \rangle \psi\}$$

The following leaf is built in the first step of the algorithm.

$$\Gamma_0 = (\{p_2, c, \langle \overline{2} \rangle \top\}, \varnothing, \varnothing)$$

In the second step, we obtain the following subtrees.

$$\Gamma_1 = (\{p_2, c, \langle 2 \rangle \top, \langle \overline{1} \rangle \top, \langle 2 \rangle \psi\}, \varnothing, \Gamma_0)$$
$$\Gamma_2 = (\{p_2, c, \langle 2 \rangle \top, \langle \overline{2} \rangle \top, \langle 2 \rangle \psi\}, \varnothing, \Gamma_0)$$

A new tree with Γ_1 and Γ_2 as subtrees is formed in step 3.

$$\Gamma_3 = (\{p_2, c, \langle 1 \rangle \top, \langle 2 \rangle \top, \langle 1 \rangle \psi, \langle 2 \rangle \psi, \langle \overline{1} \rangle \top\}, \Gamma_1, \Gamma_2)$$

We conclude in step 4 by finding the following satisfying triple, which corresponds to the model of Figure 4.6.

$$(\{p_1, \langle 1 \rangle \top, \langle 1 \rangle \psi, \langle 1 \rangle \langle (1|2)^\star \rangle p_2^{\geq 4}\}, \Gamma_3, \varnothing)$$

4.3 Correctness and Complexity

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 new trees.

In this Section, we show that the algorithm is sound and complete: given a formula, it is satisfiable if and only if the algorithm returns 1.

We conclude with an analysis of the complexity of the algorithm.

4.3.1 Soundness

If the algorithm terminates with a candidate tree, we show that the initial formula is satisfiable.

We prove soundness similarly as in Section 3.4.1. First we build a tree structure out of the ϕtree Γ that makes the algorithm to stop. Such a tree is defined by $K(\Gamma)$ (Definition 3.4.1).

Theorem 4.3.1 (Soundness)**.** *Given a formula ϕ, if the satisfiability algorithm returns* 1*, due to $\Gamma \Vdash \phi$ ($\Gamma \in ST$), then $K(\Gamma)$ satisfies ϕ.*

Proof. We proceed by induction on the structure of ϕ. All cases, but the one of counting formulas, are exactly the same as the ones in the proof of Theorem 3.4.1.

We now consider the case of counting formulas $\langle\alpha\rangle\psi^{\#k}$. In the greater than case $>$, we rely on the $k+1$ selected nodes that have to satisfy $\psi \wedge c$ thus ψ. In addition, in the less than case $\leq$, every node that is not counted has to satisfy $\neg\psi \wedge \neg c$, so in particular $\neg\psi$. In both cases we conclude by induction. □

4.3.2 Completeness

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

Assume that a formula ϕ is satisfiable by a tree K. 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 start by annotating counted nodes along with their corresponding counting proposition, yielding a new tree K_c.

Definition 4.3.1. Given a formula ϕ satisfied by a smallest tree K at node $n^\star$, we annotate nodes counted by counting subformulas of ϕ, yielding a new tree K_c as follows.

- The nodes and shapes of the trees are the same: $\mathcal{N}^K = \mathcal{N}^{K_c}$, and $\mathcal{R}^K = \mathcal{R}^{K_c}$.
- By annotate, we mean enriching the labelling function of K_c, such that it labels the counted nodes with their corresponding counting proposition. Such counting propositions are special propositions that can occur in the same node with other propositions.
  - We first copy the labelling function of K: for every node n, if $\mathcal{L}^K(n) = p$, then $\mathcal{L}^{K_c}(n) = p$.
  - The annotation procedure then starts from $n^\star$ and by induction on ϕ, 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, we recursively annotate from the node under the modality. For $\langle\alpha\rangle\psi^{\#k}$, we annotate every counted node with a fresh counting proposition corresponding to the formula: for every node n such that $n \in [\![\psi]\!]^K_V$, we have that $\mathcal{L}^{K_c}(n) = c$. For $\langle\alpha\rangle\psi^{>k}$, we annotate exactly $k+1$ nodes.

We now show K and K_c are equivalent.

Lemma 4.3.1. *A formula is satisfied by a tree K if and only if it is also satisfied by K_c.*

Figure 4.12 Formula induced by a lean

$$\frac{\psi \in lean(\phi)}{\psi \mathrel{\dot\in} lean(\phi)} \qquad \frac{\psi_1 \mathrel{\dot\in} lean(\phi) \quad \psi_2 \mathrel{\dot\in} lean(\phi)}{\psi_1 \wedge \psi_2 \mathrel{\dot\in} lean(\phi)} \qquad \frac{\psi_i \mathrel{\dot\in} lean(\phi)}{\psi_1 \vee \psi_2 \mathrel{\dot\in} lean(\phi)}$$

$$\frac{}{\top \mathrel{\dot\in} lean(\phi)} \qquad \frac{\psi \notin lean(\phi)}{\neg\psi \mathrel{\dot\in} lean(\phi)}$$

Proof. We proceed by recursion on the derivation $n^\star \in [\![\phi]\!]^K_\varnothing$. 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 $\langle\alpha\rangle\psi^{>k}$, as there are exactly $k+1$ nodes annotated, the property is true by induction. For $\langle\alpha\rangle\psi^{\leq k}$, 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 $[\![\psi]\!]^K_V = [\![\psi]\!]^K_V$ and $[\![\neg\psi]\!]^K_V = [\![\neg\psi]\!]^K_V$. □

We now reuse the Definition 3.4.2 of lean labelled trees $\Gamma(K_c)$, which will be called simply Γ in the remainder of the Subsection.

We now 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 $\psi \mathrel{\dot\in} lean(\phi)$, if it consists of the boolean combination of subformulas from the lean as defined in Figure 4.12.

Lemma 4.3.2. *Let $\langle m\rangle\psi$ be a formula in $lean(\phi)$, and let ψ' be ψ after unfolding its fixpoint formulas not under modalities. We have $\psi' \mathrel{\dot\in} lean(\phi)$.*

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

Lemma 4.3.3. *Let ψ be a formula induced by $lean(\phi)$. We have $n \in [\![\psi]\!]^K_\varnothing$ if and only if $n^\phi \vdash \psi$.*

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 4.3.4. *Let n_1 and n_2, such that $\mathcal{R}(n_1, m) = n_2$, with $m \in \{1, 2\}$. We have $\Delta_m(n_1^\phi, n_2^\phi)$.*

Proof. Let $\langle m\rangle\psi$ be a formula in $lean(\phi)$. We show that $\langle m\rangle\psi \in n_1^\phi \iff n_2^\phi \vdash \psi$. We have $\langle m\rangle\psi \in n_1^\phi$ if and only if $n_1 \in [\![\langle m\rangle\psi]\!]^K_\varnothing$ by definition of n_1^ϕ, which in turn holds if and only if $n_2 = \mathcal{R}(n_1, m) \in [\![\psi]\!]^K_\varnothing$. We now consider ψ' which is ψ after unfolding its fixpoint formulas not under modalities. We have $[\![\psi']\!]^K_\varnothing = [\![\psi]\!]^K_\varnothing$ and we conclude by Lemmas 4.3.2 and 4.3.3. □

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

Lemma 4.3.5. *Let ϕ_s be a subformula of ϕ, and ρ be a path from the root in K such that $K(\rho) \in [\![\phi_s]\!]^K_\varnothing$. We then have $\Gamma(\rho) \vdash \phi_s$.*

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 after unfolding its fixpoint formulas not under modalities. We have $[\![\phi'_s]\!]^K_\varnothing = [\![\phi_s]\!]^K_\varnothing$ and $\phi'_s \mathrel{\dot\in} lean(\phi)$. We conclude by Lemma 4.3.3.

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

For $\langle\alpha\rangle\psi^{>k}$, we have by induction for every counted node $\Gamma(\rho\rho') \vdash \psi$ and $\Gamma(\rho\rho') \vdash c$. We conclude by the conjunction rule and by the counting rule of Figure 4.9.

For $\langle\alpha\rangle\psi^{\leq k}$, we proceed as above for the counted nodes. For the nodes that are not counted, we have $\Gamma(\rho\rho') \vdash \neg\psi$ by Lemma 4.3.3 (since $\neg\psi \mathrel{\dot\in} lean(\phi)$). We conclude by remarking that the node is not annotated by c, hence $\Gamma(\rho\rho') \vdash \neg c$. □

In order to show the algorithm can build Γ, we first need to show there is a bound on the number of identical nodes occurring in a path from the root to a leaf. We proceed in two steps: first we show that counted nodes (with counted propositions) imply a bound on the number of identical ϕ-nodes on a branch for a smallest tree. Second, we show that this minimal marking is bound by $maxK(\phi)$.

We recall that ϕ is a satisfiable formula and K_c is a smallest tree such that ϕ is satisfied, and $n^\star$ is a witness of satisfiability. In the following, we call counted nodes and node $n^\star$ *annotations*.

Definition 4.3.2 (Projection)**.** 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 4.3.6. *Let K_c be the annotated tree satisfying a formula ϕ, 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 consider the tree K_s where n_2 is "grafted" upon n_1. Formally, let ρ_1 be the path to n_1 and $\rho_1\rho_2$ the path to n_2. We remove every node whose path is of the form $\rho_1\rho_3$ where ρ_2 is not a prefix of ρ_3, and we also remove node n_2. The mapping $\mathcal{R}^{K_s}$ from nodes and modalities to nodes is the same as before for the node that are kept except for n_1, where $\mathcal{R}^{K_s}(n_1, 1) = \mathcal{R}^{K_c}(n_2, 1)$ and

$\mathcal{R}^{K_s}(n_1, 2) = \mathcal{R}^{K_c}(n_2, 2)$. 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, 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 $\Gamma_s(\rho_1\rho_3) = \Gamma(\rho')$.

We now show that Γ_s still satisfies ϕ at $n^\star$, a contradiction since this tree is strictly smaller than Γ.

First, as there was no annotation projected, $n^\star$ is still part of this tree at a path ρ_s. We show that we have $\Gamma_s(\rho_s) \vdash \phi$ by induction on the derivation $\Gamma(\rho) \vdash \phi$. Let $\Gamma(\rho') \vdash \phi'$ 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.

We now turn to the modality case, $\langle m\rangle\phi'$ where ϕ' contains a counting formula. If ρ' is neither ρ_1 nor $\rho_1\rho_2$, we deduce from the fact that ρ'_s is defined that $(\rho' m)_s$ is also defined and we conclude by induction. We now assume that ρ' is either ρ_1 or $\rho_1\rho_2$ and find a contradiction. First, remark that $\Gamma(\rho') \vdash \langle m\rangle\phi'$ implies that the navigation generated by $\langle m\rangle\phi'$ is in $\Gamma(\rho_1) = \Gamma(\rho_1\rho_2)$. As each syntactic occurrence of a counting formula mentions a distinct counting proposition c, this is possible only if the counting formula is under a fixpoint or under another counting formula, both of which are impossible.

We finally turn to the counting case $\langle\alpha\rangle\psi^{\#k}$. 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 show that the remaining nodes reached through a crossover remain reachable (there cannot be any counted node in the part of the tree that is removed since counted nodes are annotated and there was no annotation in the part removed). 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 an instance of the trail, thus the same node is still reached.

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 $\alpha^\star_{r_i}$ 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, $\rho_p\rho_s \in \alpha_1, \ldots, \alpha_i$, and 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.) If there are several satisfying indices i, we consider the smallest.

We first show that a counting proposition is necessarily mentioned in a formula of n_2^ϕ, by contradiction. Assume no counting proposition is mentioned, yet the counting crossed-over. This can only occur for a "less than" counting formula that reaches n_2 which is not counted (because the formula was false), and if there is no path whose ρ_n is a strict prefix that is an instance of α (otherwise, by definition of the lean and of nav (Figure 4.7), a formula of the form $\mathrm{nav}(\alpha', (\psi \wedge c) \vee (\neg\psi \wedge \neg c))$ would be true and thus would be present, contradicting the assumption that no counting proposition is mentioned). Since

$n_1^\phi = n_2^\phi$, the same is true for n_1^ϕ, a direct contradiction to the fact that n_2 is also reached by the trail. Thus counting propositions are mentioned in n_1^ϕ and n_2^ϕ.

We next 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, i.e., neither i nor j may be $n+1$. Recall that α_{n+1} does not contain a repeated subtrail. Thus every formula of n_2^ϕ mentioning c is of the form $\mathrm{nav}(\alpha', \psi)$, where α' does not contain a repetition. We consider the largest such formula. Since n_1 is before n_2 in the path from the counting node to the counted node, a similar formula with a larger trail or with a repetition must occur in n_1^ϕ, contradicting $n_1^\phi = n_2^\phi$.

Consider next 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 subtrail). 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 $(\rho_s^1|\rho_s^2)\alpha_i \dots \alpha_{n+1}$, 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 $\langle\alpha\rangle\psi^{\leq k}$, 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 $\Gamma(\rho'\rho_n) \vdash \neg\psi$ 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 1, then we build the smaller tree by doing $\mathcal{R}^{K_s}(n_1, 1) = \mathcal{R}^{K_c}(n_2, 1)$ (we remove the 2 subtree from n_2 instead of n_1). Symmetrically, if the first modality is a 2, we consider $\mathcal{R}^{K_s}(n_1, 2) = \mathcal{R}^{K_c}(n_2, 2)$ as smaller tree. The rest of the proof proceeds as above. □

We are now ready to show the algorithm consistently builds Γ.

Theorem 4.3.2 (Completeness)**.** *Given a smallest model K of a formula ϕ, the algorithm constructs the satisfying tree $\Gamma(K_c)$.*

Proof. We proceed by induction on the height of K.

Since leaves are produced by the algorithm in the first step, the base case is immediate.

Let K_1 and K_2 the two main subtrees of K_c. By induction we know the trees $\Gamma(K_i)$ ($i = 1, 2$) are already in ST.

That $\mathcal{R}^{K_c}$ implies Δ comes from Lemma 4.3.4.

We now show the root node is available to be joined, by means of Δ, with $\Gamma(K_i)$. By Lemma 4.3.6, we know there are at most $k+1$ identical nodes, where k is the number of annotations, in every path from the root to a leaf. The number of annotations is l, where l is the number of counted nodes. We show by immediate induction on the formula ϕ that l is bound by $maxK(\phi)$,

as defined in Figure 4.11. We conclude by remarking that $maxK(\phi)+2$ is the number of identical nodes we allow in the algorithm. □

4.3.3 Complexity

We now show that the time complexity of the satisfiability algorithm is exponential with respect to the formula size. This is achieved in two steps: we first show that the lean size is linear in the formula size, then we show that the algorithm has a single exponential complexity with relation to the lean size.

Lemma 4.3.7. *The lean size is linear in terms of the original formula size.*

Proof Sketch. First note that the size of the lean is the number of elements it contains; the size of each element does not matter.

It was shown in [Genevès et al., 2007] that the size of the lean generated by a non-counting formula is linear with respect to the formula size.

We now describe the case for counting formulas. The lean consists of propositions and of modal subformulas, including the ones generated by the navigation of counting formulas (Figure 4.7). Moreover, each counting formula adds one fresh counting proposition. In the case of "less than" formulas $\langle\alpha\rangle\psi^{\leq k}$, a duplication occurs due to the consideration of the negated normal form of ψ. Since there is no counting under counting, this duplication and the fact that the negated normal form of a formula is linear in the size of the original formula result in the lean remaining linear. Another duplication occurs in the case of counting formulas of the form $\langle\alpha_1|\alpha_2\rangle\psi^{\#k}$. This duplication does not double the size of the lean, however, since ψ still occurs only once in the lean, thus the number of elements in the lean induced by $\text{nav}(\alpha_1,\psi)\vee\text{nav}(\alpha_2,\psi)$ is the same as the sum of the ones in $\text{nav}(\alpha_1,\psi)$ and in $\text{nav}(\alpha_2,\cdot)$. □

Theorem 4.3.3. *The satisfiability algorithm for the logic is decidable in time* $2^{O(n)}$*, where* n *is the size of the lean.*

Proof Sketch. The maximum number of considered nodes is the number of distinct tree nodes which is 2^n, the number of subsets of the lean. For a given formula ϕ, the number of occurrences of the same node in the tree is bounded by $maxK(\phi)\leq k*m$, where k is the greatest constant occurring in the counting formulas and m is the number of counting subformulas of ϕ. Hence the number of steps of the algorithm is bounded by 2^n*k*m.

At each iteration, the main operation performed by the algorithm is the composition of trees stored in AUX. The cost of each iteration consists in: the different searches needed to form the necessary triples (n,Γ_1,Γ_2), the nmax function and Δ. Since the total number of nodes is exponential, and the number of different subtrees too, therefore the maximum number of newly formed trees (triples) at each step has also an exponential bound. The function nmax performs a single traversal of the tree which is also exponential. Since the entailment relation involved in the definition of Δ is local, Δ is performed in

linear time. Computing the containment $AUX \subseteq ST$ and the union $ST \cup AUX$ are linear operations over sets of exponential size.

The stop condition of the algorithm is checked by the global entailment relation. It involves traversals parametrized by the number of trees, the number of nodes in each tree, the number of traversals for the entailment relation of counting formulas, and $maxK(\phi)$. Its time complexity is bounded by $(2^n * k * m)^3$.

Hence, the total time complexity of the algorithm is bounded by $(2^n * k * m)^{k'}$, for some constant k'. □

4.4 Application to XML

We now show the logic can be used as a reasoning framework for XPath decision problems in the presence of XML types, where expressions can have counting constraints.

Counting constraints on both XPath expressions and XML Types have been already considered in Sections 3.5 and 3.6, respectively. Such constraints can only be imposed on children nodes. In this Section, the navigation involved in the counting constraints can be any arbitrary XPath expression, that is, a multi-directional and recursive path.

4.4.1 XPath

We have already introduced an extension of Core XPath with counting constraints on children in Section 3.6. An example of a XPath expressions with counting on children is the following.

/company/personnel[count(employee)> 10000]/name

Such expression selects the name of companies with more than 10000 employees. The node of employees are assumed to be children nodes of the personnel nodes.

We also show in Section 3.6 how to use graded modalities of μ-calculus to model the counting constraints. For instance, the following expressions are equivalent.

$$\mathbf{ancestor::p_1}[\mathit{count}(\text{child} :: p_2) \leq 5]$$
$$(\mathbf{p_1} \wedge \langle \mathbf{1} \rangle \mu \mathbf{x}.\top \vee \langle \mathbf{1} \rangle \mathbf{x} \vee \langle \mathbf{2} \rangle \mathbf{x}) \wedge p_2^{\leq 5}$$

The XPath specification allows counting on other nodes than children nodes. In particular, it is allowed to count on nodes reachable by means of an arbitrary XPath expressions. For instance, consider the following expression.

/company[count(descendant/employee)> 10000]/name

We now extend the Core XPath language with counting constraints defined in Figure 4.13. The interpretation of XPath expressions is formally defined in Figure 3.16.

Figure 4.13 Syntax of Core XPath expressions

Axis	:=	self | child | parent | descendant | ancestor |
		following-sibling | preceding-sibling | following | preceding
NameTest	:=	QName | $*$
Step	:=	Axis :: NameTest
PathExpr	:=	PathExpr/PathExpr | PathExpr[Qualifier] | Step
Qualifier	:=	PathExpr | Qualifier and Qualifier | Qualifier or Qualifier |
		not Qualifier | CountExpr | @n
QualifierC	:=	PathExprC | QualifierC and QualifierC | QualifierC or QualifierC |
		not QualifierC | @n
PathExprC	:=	PathExprC/PathExprC | PathExprC[QualifierC] | Step
CountExpr	:=	$count$(PathExprC)$\#k$
#	:=	$\leq \mid > \mid =$
XPathExpr	:=	PathExpr | /PathExpr | XPathExpr union PathExpr |
		XPathExpr intersect PathExpr |
		XPathExpr except PathExpr

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:

$$\psi = a \wedge \mu x.\langle\overline{1}\rangle\top \vee \langle\overline{2}\rangle x$$

The logical translation of the above XPath expression is:

$$\psi \wedge \langle 1\rangle\langle(1|2)^{\star}\rangle(b \wedge \mu x.\langle\overline{1}\rangle c \vee \langle\overline{2}\rangle x)^{>5}$$

This formula holds for nodes selected by the XPath expression.
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.

Figure 4.14 Translation of XPath counting expressions into graded paths

$$\langle count(P)\rangle^q_\chi = @\mathrm{n} \wedge \langle(\overline{1}|\overline{2})^\star(1|2)^\star\rangle(\langle P\rangle_{\chi\wedge@\mathrm{n}})^{\#k}$$

The first path can be translated in the logic as follows:

$$\vartheta = b \wedge \mu x.\langle\overline{1}\rangle(a \wedge \mu y.\langle\overline{1}\rangle\top \vee \langle\overline{2}\rangle y) \vee \langle\overline{2}\rangle x$$

The counting part 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:

$$\vartheta \wedge @\mathrm{n} \wedge \langle(\overline{1}|\overline{2})^\star, (1|2)^\star\rangle\eta^{>3}$$

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:

$$\eta = h \wedge \mu x.\langle\overline{1}\rangle(e \wedge \mu y.\langle\overline{1}\rangle@\mathrm{n} \vee \langle\overline{2}\rangle y) \vee \langle\overline{2}\rangle x \vee \langle\overline{1}\rangle x$$

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 $(\overline{1}|\overline{2})^\star(1|2)^\star$ 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.

Note the only difference of the XPath expressions defined in Figure 4.13 with respect to the expressions defined in Figure 3.15 is that counting expressions in Figure 4.13 allow arbitrary XPath expressions with no nested counting. Hence, we reuse the definition of the translation defined in Figure 3.18. and we only update the translation of counting expressions occurring in qualifiers (Figure 4.14).

Lemma 4.4.1. *The translation of Core XPath expressions with counting constraints into the logic is linear.*

It is proven by structural induction in a similar manner as done in Section 3.5 or by [Genevès et al., 2007] (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.

4.4.2 XML Types

Regular Tree Types, defined in Section 3.6, capture most of the schemas in use today [Murata et al., 2005]. The logic can express all regular tree languages (it is easy to prove that regular expression types in the manner of e.g., [Hosoya et al., 2005] can be linearly translated into the logic: see Section 3.6 or [Genevès et al., 2007]).

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

$$\phi \wedge \langle 1 \rangle (\langle 2^\star \rangle b^{>3} \wedge \langle 2^\star \rangle b^{\leq 9})$$

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

$$\begin{aligned} \phi = \quad & (a \wedge (\neg\langle 1 \rangle \top \vee \langle 1 \rangle \psi)) \wedge \neg\langle 2 \rangle \top \\ \psi = \quad & \mu x.\, (b \wedge \neg\langle 1 \rangle \top \wedge \neg\langle 2 \rangle \top) \vee (b \wedge \neg\langle 1 \rangle \top \wedge \langle 2 \rangle x) \end{aligned}$$

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:

$$p \wedge \langle (\overline{1}|\overline{2})^\star, \overline{1} \rangle ul^{>3}$$

We now conclude with a logic-based reasoning framework for the XML decision problems defined in Subsection 3.6.3.

Theorem 4.4.1. *In the context of XPath and XML schema languages with cardinality constraints, the XML decision problems of containment, equivalence, emptiness, overlap and coverage, are decidable in exponential time with respect to the problem size.*

4.5 Conclusion

In this Chapter we have extended the alternation-free μ-calculus with converse for finite trees with graded paths. From a given context, graded paths allow describing numerical constraints on nodes reachable by some regular paths. The novelty of this logic is that paths can be multidirectional and recursive such as those obtained by the composition of child, parent, descendant, and ancestor XPath axes.

A tableau-based satisfiability algorithm for the logic is presented and proved correct. The computational complexity of the algorithm is shown to be in exponential time with respect to the original formula size.

We also introduced a linear translation of XPath expressions into the logic. The counting features of this logic allow capturing the counting constraints of arbitrary XPath expressions except those involving nesting. We also showed, through examples, the potential benefits of counting deeply in XML schema languages by means of conjunctions of schemas and counting path expressions. All traditional decision problems of XPath expressions such as containment, equivalence and coverage are solved in exponential time by means of the satisfiability decision procedure presented here.

The logic presented in this chapter is more general than the fully enriched μ-calculus of Chapter 3. However, we syntactically disallow for the graded paths the nesting of counting and counting under recursion. Therefore, there are still some kinds of counting constraints that cannot be expressed by this logic in a succinct manner.

Chapter 5

Conclusions and Perspectives

We now summarize the main contributions of this work, and propose further research directions.

5.1 Summary of Contributions

The main contributions of this thesis are twofold: first, we showed that the fully enriched μ-calculus for trees is decidable unlike its graph counterpart. The fully enriched μ-calculus corresponds to μ-calculus equipped with converse programs, graded modalities, and nominals. Graded modalities are special modal operators able to express numerical constraints on the number children nodes. Second, we designed a tree logic with graded paths. Graded paths allow the specification of cardinality constraints on nodes that are "deeper" in the tree beyond the children nodes.

5.1.1 Graded Modalities

[Bonatti and Peron, 2004] showed the undecidability of fully enriched μ-calculus. We showed in Chapter 3 that the fully enriched μ-calculus is decidable when interpreted on finite trees. More precisely, we developed and proved correct a tableau-based satisfiability algorithm for the logic. The computational complexity of the algorithm is exponential with respect to the formula size, even when the numbers occurring in the formulas are coded in binary.

[Dal-Zilio et al., 2004] and [Seidl et al., 2004] developed a similar tree logic able to express Presburger constraints on children nodes. These constraints are expressed via formulas of Presburger arithmetic, which are more general than the numerical restrictions introduced here for graded modalities. Compared to the the fully enriched μ-calculus presented here, the work of

[Dal-Zilio et al., 2004] and [Seidl et al., 2004] do not support backward navigation even between sibling nodes nor deep counting.

In Chapter 3, we presented a linear translation of XML types and XPath expressions with counting operators into the logic. This translation allows formulating succinctly typical problems on XPath queries such as satisfiability, type checking under regular tree types constraints, containment, or equivalence. The main navigational features of XPath, usually called Core XPath are captured by the logic: recursive navigation (descendant axis), backward navigation (parent and ancestor axis), qualifiers, and counting operators over children nodes (child axis).

5.1.2 Graded Paths

In Chapter 4, we extended the μ-calculus with converse modalities and nominals with graded paths. In contrast with the logic of Chapter 3, counting operators involved in the graded paths are restricted not to occur under the scope of fixpoints or other counting operators.

[Demri and Lugiez, 2006] extended the logic of [Dal-Zilio et al., 2004] with Presburger constraints, by means of regular forward paths, on nodes with a scope that goes beyond children nodes. They proved that such a logic is undecidable. In the logic of Chapter 4, we chose a different tradeoff: comparisons are restricted to constants but applied to nodes accessible by regular paths. The logic with graded paths presented here supports compositions of recursive forward and backward navigation in the same path and for both counting and non-counting ones. This is achieved in formulas by means of fixpoint operators and the forward and converse modalities.

With the notable exception of [Demri and Lugiez, 2006], the computational machinery used for reasoning is automata-based. Here, we use tableaux satisfiability algorithms instead.

We have shown in Chapter 4 that the logic can linearly embed Core XPath expressions and XML types enhanced with counting operators over regular and multi-directional paths. Except the restriction that forbids the nesting of counting operators, in contrast with the XPath fragment supported in Chapter 3, the counting operators proposed in Chapter 4 are sufficient to capture all the counting paths expressed by the XPath language (recursive and multi-directional).

5.2 Perspectives

There are a number of interesting and promising directions for further research that builds on the results and ideas developed in this dissertation.

Typical further research directions for this work are the extensions of tree logics with operators in order to increase their expressivity or (and) succinctness. Novel implementation techniques for these enriched logics are needed, including algorithms that do not always perform in the worst case. Such ex-

tensions may be motivated by practical problems not only in theory of formal languages, but in other areas such as the verification of software and hardware.

5.2.1 Fully Graded Paths

In order to show completeness of the satisfiability algorithm of μ-calculus with graded paths in Chapter 4, we restricted counting operators not to occur in scope of fixpoints or other counting operators. It remains an open question to check if this restriction can be eliminated. This can result in a much simpler syntax for the logic and the support of XPath nested counting operators.

5.2.2 Data Values

Data values comparisons for node and attributes values in XML documents are commonly used in XPath experessions, XSLT transformations and schema languages such as Schematron. However, preliminary results reported in the literature have shown that reasoning with data values can be very hard. In particular, [Bojanczyk et al., 2009] characterized XPath fragments including data values, in the context of FO^2, and the complexity results are shown to be ranging from NEXPTIME to 3NEXPTIME.

In the modal setting, [Jurdzinski and Lazic, 2007] identify some decidable fragments of XPath with data equality test via an extension of μ-calculus for trees. However, the decision algorithm presented there turned out to have a non-primitive recursive computational cost.

A further research direction, in the setting of data trees, is the development of decision algorithms with the logic techniques presented in this work. The presence of both, data values and counting constraints, in reasoning frameworks for trees has not been well-studied neither.

5.2.3 Implementations

A direct implementation of the satisfiability algorithm presented in Chapter 4 is underway. [Genevès et al., 2007] and [Tanabe et al., 2005] have shown that tableau-based algorithms for μ-calculus can be efficiently implemented and take full advantage of symbolic techniques such as Binary Decision Diagrams. They have also shown that such solvers can be used successfully in practical systems to solve realistic and even large use cases. The satisfiability algorithms developed in this thesis requires more investigations to reach the same level of maturity. In particular, it is interesting to explore ways of combining such solvers with satisfiability modulo solvers (SMT) to obtain the same type of monotonic processes that allows to seek witnessing trees while propagating the numerical constraints at the same time. It is also interesting to have an idea of the orders of magnitudes in terms of time and memory space required to solve XML static analysis problems with counting for realistic cases. In particular, this can provide figures for some important applications such

as those described in [Genevès et al., 2009], [Genevès and Layaïda, 2010a] and [Layaïda and Genevès, 2010].

5.2.4 Modular Regular Tree Types

One of the advantages of the logic developed in this thesis is its ability to capture both paths and types. One possible use of this logic is to develop schema languages that uses both types and path expressions. It is well known that one of the drawbacks of schema languages such as XML Schema and RelaxNG is their reuse. The reason behind this lies in the fact that usually schema designers describe the types in the forward direction starting from the root and by declaring the content models of every sub-schema fragments. If the type of these fragments is sensitive to the context, the context information is propagated across the different elements nesting. This process yields schemas and sub-schemas that can be hardly reused.

An interesting alternative is to design new schema languages where the context is captured by means of backward path expressions declared from within the sub-schema. Such languages are more modular in the sense that schema fragments can be extracted and re-used in other contexts while retaining their contextual constraints. This will help also building some kind of modular schema libraries that can be used as building blocks for more involved ones.

5.2.5 Verification of Programs

Applications of this work can be found in other areas such as program verification of linked data structures. Proving correctness of programs is a crucial part in the verification of software, such as operating or real-time systems. The implementation of efficient high level program structures are often based on balanced tree structures, such as AVL trees, red-black trees, splay trees, etc.

Reasoning frameworks with in-depth counting constraints, such as the ones handled in Chapter 4, play a major role in the verification of balanced tree structures, as presented by [Habermehl et al., 2010] and [Manna et al., 2007].

Therefore, we believe it is possible to extend the field of applications of the reasoning frameworks developed in this work to the verification of balanced tree structures.

Bibliography

[Afanasiev et al., 2005] Afanasiev, L., Blackburn, P., Dimitriou, I., Gaiffe, B., Goris, E., Marx, M., and de Rijke, M. (2005). PDL for ordered trees. *Journal of Applied Non-Classical Logics*, 15(2):115–135. 30, 34, 73

[Alur and Henzinger, 1989] Alur, R. and Henzinger, T. A. (1989). A really temporal logic. In *FOCS*, pages 164–169. IEEE. 31

[Arnold and Niwinski, 1992] Arnold, A. and Niwinski, D. (1992). Fixed point characterization of weak monadic logic definable sets of trees. In Nivat, M. and Podelski, A., editors, *Tree Automata and Languages*, pages 159–188. North-Holland. 30

[Bárány et al., 2009] Bárány, V., Kaiser, L., and Rabinovich, A. (2009). Cardinality quantifiers in MLO over trees. In Grädel, E. and Kahle, R., editors, *CSL*, volume 5771 of *Lecture Notes in Computer Science*, pages 117–131. Springer. 27

[Barceló and Libkin, 2005] Barceló, P. and Libkin, L. (2005). Temporal logics over unranked trees. In *LICS*, pages 31–40. 29

[Beltyukov, 1976] Beltyukov, A. P. (1976). Decidability of the universal theory of natural numbers with addition and divisibility. *Zapiski Nauch. Sem. Leningrad Otdeleniya Mathematical Institute*, 60:15–28. 24

[Benedikt et al., 2003] Benedikt, M., Fan, W., and Kuper, G. M. (2003). Structural properties of XPath fragments. In Calvanese, D., Lenzerini, M., and Motwani, R., editors, *ICDT*, volume 2572 of *Lecture Notes in Computer Science*, pages 79–95. Springer. 25

[Bés, 2002] Bés, A. (2002). A survey of arithmetical definability. In *A Tribute to Maurice Boffa. Bulletin de la Société Mathématique de Belgique*. 24

[Bianco et al., 2009] Bianco, A., Mogavero, F., and Murano, A. (2009). Graded computation tree logic. In *LICS*, pages 342–351. IEEE Computer Society. 33, 68, 69

[Biehl et al., 1996] Biehl, M., Klarlund, N., and Rauhe, T. (1996). Mona: decidable arithmetic in practice (demo). In *Formal Techniques in Real-Time and Fault-Tolerant Systems, 4th International Symposium, Uppsala, LNCS 1135*. 27

[Bojanczyk et al., 2009] Bojanczyk, M., Muscholl, A., Schwentick, T., and Segoufin, L. (2009). Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3). 25, 95

[Bonatti et al., 2006] Bonatti, P. A., Lutz, C., Murano, A., and Vardi, M. Y. (2006). The complexity of enriched μ-calculi. In Bugliesi, M., Preneel, B., Sassone, V., and Wegener, I., editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 540–551. Springer. 32, 37, 56, 69

[Bonatti and Peron, 2004] Bonatti, P. A. and Peron, A. (2004). On the undecidability of logics with converse, nominals, recursion and counting. *Artif. Intell.*, 158(1):75–96. 21, 32, 37, 93

[Bozga and Iosif, 2005] Bozga, M. and Iosif, R. (2005). On decidability within the arithmetic of addition and divisibility. In Sassone, V., editor, *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 425–439. Springer. 24

[Bray et al., 2004] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2004). Extensible markup language (XML). W3C recommendation. 63

[Bryant, 1986] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691. 27

[Büchi, 1960] Büchi, J. R. (1960). Weak second-order logic and finite automata. *S. Math. Logik Gründlagen Math*, 6:66–92. 15, 21, 26, 27

[Büchi, 1962] Büchi, J. R. (1962). On a decision method in restricted second order arithmetic. In *Proc. Int. Congr. for Logic Methodology and Philosophy of Science*, pages 1–11. Stanford University Press. 27

[Calvanese et al., 2009] Calvanese, D., Giacomo, G. D., Lenzerini, M., and Vardi, M. Y. (2009). An automata-theoretic approach to regular XPath. In Gardner, P. and Geerts, F., editors, *DBPL*, volume 5708 of *Lecture Notes in Computer Science*, pages 18–35. Springer. 31

[Cardelli and Gordon, 2000] Cardelli, L. and Gordon, A. D. (2000). Anytime, anywhere: Modal logics for mobile ambients. In *POPL*, pages 365–377. 34

[Church, 1936a] Church, A. (1936a). A note on the Entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41. 24

[Church, 1936b] Church, A. (1936b). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363. 24

[Church, 1957] Church, A. (1957). Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume 1, pages 3–50. 26

[Clark and DeRose, 1999] Clark, J. and DeRose, S. (November 1999). XML path language (XPath) version 1.0. W3C recommendation. 58

[Clark and Murata, 2001] Clark, J. and Murata, M. (2001). RELAX NG specification. OASIS committee specification. 63

[Clarke and Emerson, 1981] Clarke, E. M. and Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching-time temporal logic. In Kozen, D., editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer. 15, 29

[Colazzo et al., 2009] Colazzo, D., Ghelli, G., and Sartiani, C. (2009). Efficient asymmetric inclusion between regular expression types. In Fagin, R., editor, *ICDT*, volume 361 of *ACM International Conference Proceeding Series*, pages 174–182. ACM. 17

[Courcelle, 1990] Courcelle, B. (1990). The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75. 27

[Dal-Zilio and Lugiez, 2003] Dal-Zilio, S. and Lugiez, D. (2003). XML schema, tree logic and sheaves automata. In Nieuwenhuis, R., editor, *RTA*, volume 2706 of *Lecture Notes in Computer Science*, pages 246–263. Springer. 34

[Dal-Zilio et al., 2004] Dal-Zilio, S., Lugiez, D., and Meyssonnier, C. (2004). A logic you can count on. In Jones, N. D. and Leroy, X., editors, *POPL*, pages 135–146. ACM. 24, 34, 35, 93, 94

[Demri and Lazic, 2006] Demri, S. and Lazic, R. (2006). LTL with the freeze quantifier and register automata. In *LICS*, pages 17–26. IEEE Computer Society. 31

[Demri and Lugiez, 2006] Demri, S. and Lugiez, D. (2006). Presburger modal logic is PSPACE-Complete. In Furbach, U. and Shankar, N., editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 541–556. Springer. 20, 21, 22, 34, 35, 94

[Doner, 1970] Doner, J. (1970). Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451. 27

[Elgaard et al., 1998] Elgaard, J., Klarlund, N., and Møller, A. (1998). MONA 1.x: new techniques for WS1S and WS2S. In *Proc. 10th International Conference on Computer-Aided Verification, CAV '98*, volume 1427 of *LNCS*, pages 516–520. Springer-Verlag. 27

[Elgot, 1960] Elgot, C. C. (1960). Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51. 15, 26, 27

[Emerson and Halpern, 1986] Emerson, E. A. and Halpern, J. Y. (1986). "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178. 29

[Emerson and Jutla, 1988] Emerson, E. A. and Jutla, C. S. (1988). The complexity of tree automata and logics of programs (extended abstract). In *FOCS*, pages 328–337. IEEE. 30

[Fallside and Walmsley, 2004] Fallside, D. C. and Walmsley, P. (2004). XML schema. W3C recommendation. 63

[Ferrante et al., 2009] Ferrante, A., Napoli, M., and Parente, M. (2009). Graded-CTL: Satisfiability and symbolic model checking. In Breitman, K. and Cavalcanti, A., editors, *ICFEM*, volume 5885 of *Lecture Notes in Computer Science*, pages 306–325. Springer. 33

[Fine, 1972] Fine, K. (1972). In so many possible worlds. *Notre Dame Journal of Formal Logic*, 13(4):516–520. 32

[Fischer and Ladner, 1979] Fischer, M. J. and Ladner, R. E. (1979). Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211. 29, 30

[Fischer and Rabin, 1974] Fischer, M. J. and Rabin, M. O. (1974). Super-exponential complexity of Presburger Arithmetic. In *SIAM-AMS Symposium in Applied Mahtematics*. 24

[Gelade et al., 2009a] Gelade, W., Gyssens, M., and Martens, W. (2009a). Regular expressions with counting: Weak versus strong determinism. In Královic, R. and Niwinski, D., editors, *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 369–381. Springer. 18

[Gelade et al., 2009b] Gelade, W., Martens, W., and Neven, F. (2009b). Optimizing schema languages for XML: Numerical constraints and interleaving. *SIAM J. Comput.*, 38(5):2021–2043. 17

[Genevès, 2006] Genevès, P. (2006). *Logics for XML*. PhD thesis, Institut National Polytechnique de Grenoble. 64

[Genevès and Layaïda, 2006] Genevès, P. and Layaïda, N. (2006). A system for the static analysis of XPath. *ACM Trans. Inf. Syst.*, 24(4):475–502. 31

[Genevès and Layaïda, 2007] Genevès, P. and Layaïda, N. (2007). Deciding XPath containment with MSO. *Data Knowl. Eng.*, 63(1):108–136. 27

[Genevès and Layaïda, 2010a] Genevès, P. and Layaïda, N. (2010a). Eliminating dead-code from XQuery programs. In Kramer, J., Bishop, J., Devanbu, P. T., and Uchitel, S., editors, *ICSE (2)*, pages 305–306. ACM. 96

[Genevès and Layaïda, 2010b] Genevès, P. and Layaïda, N. (2010b). XML reasoning made practical. In Li, F., Moro, M. M., Ghandeharizadeh, S., Haritsa, J. R., Weikum, G., Carey, M. J., Casati, F., Chang, E. Y., Manolescu, I., Mehrotra, S., Dayal, U., and Tsotras, V. J., editors, *ICDE*, pages 1169–1172. IEEE. 31

[Genevès et al., 2009] Genevès, P., Layaïda, N., and Quint, V. (2009). Identifying query incompatibilities with evolving XML schemas. In Hutton, G. and Tolmach, A. P., editors, *ICFP*, pages 221–230. ACM. 96

[Genevès et al., 2007] Genevès, P., Layaïda, N., and Schmitt, A. (2007). Efficient static analysis of XML paths and types. In Ferrante, J. and McKinley, K. S., editors, *PLDI*, pages 342–351. ACM. 21, 31, 37, 38, 41, 43, 46, 47, 55, 58, 59, 62, 65, 69, 75, 87, 90, 91, 95

[Genevès and Rose, 2005] Genevès, P. and Rose, K. H. (2005). Compiling XPath for streaming access policy. In Wiley, A. and King, P. R., editors, *ACM Symposium on Document Engineering*, pages 52–54. ACM. 20

[Goble, 1970] Goble, L. F. (1970). Grades of modality. *Logique et Analyse*, 13:323–334. 32

[Gödel, 1931] Gödel, K. (1931). Über formal unentscheidbare sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, 38:173–198. 24

[Gottlob et al., 2005] Gottlob, G., Koch, C., and Pichler, R. (2005). Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491. 25, 29

[Grädel et al., 1997a] Grädel, E., Kolaitis, P. G., and Vardi, M. Y. (1997a). On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69. 25

[Grädel and Otto, 1999] Grädel, E. and Otto, M. (1999). On logics with two variables. *Theor. Comput. Sci.*, 224(1-2):73–113. 25

[Grädel et al., 1997b] Grädel, E., Otto, M., and Rosen, E. (1997b). Two-variable logic with counting is decidable. In *LICS*, pages 306–317. 26

[Gruber and Holzer, 2009] Gruber, H. and Holzer, M. (2009). Tight bounds on the descriptional complexity of regular expressions. In Diekert, V. and Nowotka, D., editors, *Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, pages 276–287. Springer. 16

[Habermehl et al., 2010] Habermehl, P., Iosif, R., and Vojnar, T. (2010). Automata-based verification of programs with tree updates. *Acta Inf.*, 47(1):1–31. 96

[Henriksen et al., 1995] Henriksen, J., Jensen, J., Jørgensen, M., Klarlund, N., Paige, B., Rauhe, T., and Sandholm, A. (1995). 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*. 16, 21, 27

[Hollunder and Baader, 1991] Hollunder, B. and Baader, F. (1991). Qualifying number restrictions in concept languages. In *KR*, pages 335–346. 32

[Hosoya and Pierce, 2003] Hosoya, H. and Pierce, B. C. (2003). Xduce: A statically typed XML processing language. *ACM Trans. Internet Techn.*, 3(2):117–148. 40

[Hosoya et al., 2005] Hosoya, H., Vouillon, J., and Pierce, B. C. (2005). Regular expression types for XML. *ACM Trans. Program. Lang. Syst.*, 27(1):46–90. 91

[Janin and Walukiewicz, 1996] Janin, D. and Walukiewicz, I. (1996). On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In Montanari, U. and Sassone, V., editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer. 30

[Jensen et al., 1997] Jensen, J. L., Joergensen, M. E., Klarlund, N., and Schwartzbach, M. I. (1997). Automatic verification of pointer programs using monadic second-order logic. In *PLDI '97*. 27

[Jurdzinski and Lazic, 2007] Jurdzinski, M. and Lazic, R. (2007). Alternation-free modal mu-calculus for data trees. In *LICS*, pages 131–140. IEEE Computer Society. 32, 95

[Kieronski and Otto, 2005] Kieronski, E. and Otto, M. (2005). Small substructures and decidability issues for first-order logic with two variables. In *LICS*, pages 448–457. IEEE Computer Society. 25

[Kilpeläinen and Tuhkanen, 2003] Kilpeläinen, P. and Tuhkanen, R. (2003). Regular expressions with numerical occurrence indicators - preliminary results. In Kilpeläinen, P. and Päivinen, N., editors, *SPLST*, pages 163–173. University of Kuopio, Department of Computer Science. 17

[Kilpeläinen and Tuhkanen, 2007] Kilpeläinen, P. and Tuhkanen, R. (2007). One-unambiguity of regular expressions with numeric occurrence indicators. *Inf. Comput.*, 205(6):890–916. 17

[Kleene, 1956] Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press. 15, 16, 26

[Kozen, 1983] Kozen, D. (1983). Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354. 30

[Kracht, 1995] Kracht, M. (1995). Syntactic codes and grammar refinement. *Journal of Logic, Language and Information*, 4(1):41–60. 30

[Kripke, 1963] Kripke, S. A. (1963). A semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96. 28, 38

[Kupferman et al., 2002] Kupferman, O., Sattler, U., and Vardi, M. Y. (2002). The complexity of the graded μ-calculus. In Voronkov, A., editor, *CADE*, volume 2392 of *Lecture Notes in Computer Science*, pages 423–437. Springer. 32, 55

[Ladner, 1977] Ladner, R. E. (1977). The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6(3):467–480. 28

[Laroussinie and Schnoebelen, 2000] Laroussinie, F. and Schnoebelen, P. (2000). Specification in CTL+Past for verification in CTL. *Inf. Comput.*, 156(1-2):236–263. 29

[Layaïda and Genevès, 2010] Layaïda, N. and Genevès, P. (2010). Debugging standard document formats. In Rappa, M., Jones, P., Freire, J., and Chakrabarti, S., editors, *WWW*, pages 1269–1272. ACM. 96

[Lazic, 2006] Lazic, R. (2006). Safely freezing LTL. In Arun-Kumar, S. and Garg, N., editors, *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 381–392. Springer. 31

[Libkin and Sirangelo, 2010] Libkin, L. and Sirangelo, C. (2010). Reasoning about XML with temporal logics and automata. *J. Applied Logic*, 8(2):210–232. 31

[Lipshitz, 1976] Lipshitz, L. (1976). The diophantine problem for addition and divisibility. *Transaction of the American Mathematical Society*, 235:271–283. 24

[Manna et al., 2007] Manna, Z., Sipma, H. B., and Zhang, T. (2007). Verifying balanced trees. In Artëmov, S. N. and Nerode, A., editors, *LFCS*, volume 4514 of *Lecture Notes in Computer Science*, pages 363–378. Springer. 24, 96

[Marx and de Rijke, 2005] Marx, M. and de Rijke, M. (2005). Semantic characterizations of navigational XPath. *SIGMOD Record*, 34(2):41–46. 25

[Mendelson, 1964] Mendelson, E. (1964). *Introduction to Mathematical Logic*. Wadsworth & Brooks. 24

[Meyer, 1975] Meyer, A. R. (1975). Weak monadic second-order theory of successor is not elementary-recursive. In Parikh, R., editor, *Proceedings of Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 132–154. 28, 35

[Meyer and Stockmeyer, 1972] Meyer, A. R. and Stockmeyer, L. J. (1972). The equivalence problem for regular expressions with squaring requires exponential space. In *FOCS*, pages 125–129. IEEE. 18

[Miklau and Suciu, 2004] Miklau, G. and Suciu, D. (2004). Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45. 29

[Møller and Schwartzbach, 2001] Møller, A. and Schwartzbach, M. I. (2001). The pointer assertion logic engine. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '01*. Also in SIGPLAN Notices 36(5) (May 2001). 27

[Moller and Rabinovich, 2003] Moller, F. and Rabinovich, A. M. (2003). Counting on CTL*: on the expressive power of monadic path logic. *Inf. Comput.*, 184(1):147–159. 33

[Mortimer, 1975] Mortimer, M. (1975). On languages with two variables. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 21. 25

[Murata et al., 2005] Murata, M., Lee, D., Mani, M., and Kawaguchi, K. (2005). Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704. 17, 63, 91

[Neven, 2002] Neven, F. (2002). Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46. 40

[Nguyen et al., 2007] Nguyen, H. H., David, C., Qin, S., and Chin, W.-N. (2007). Automated verification of shape and size properties via separation logic. In Cook, B. and Podelski, A., editors, *VMCAI*, volume 4349 of *Lecture Notes in Computer Science*, pages 251–266. Springer. 24

[Oppen, 1978] Oppen, D. C. (1978). A $2^{2^{2^{pn}}}$ upper bound on the complexity of presburger arithmetic. *J. Comput. Syst. Sci.*, 16(3):323–332. 24

[Pacholski et al., 1997] Pacholski, L., Szwast, W., and Tendera, L. (1997). Complexity of two-variable logic with counting. In *LICS*, pages 318–327. 26

[Pnueli, 1977] Pnueli, A. (1977). The temporal logic of programs. In *FOCS*, pages 46–57. IEEE. 15, 28

[Pnueli, 1985] Pnueli, A. (1985). Linear and branching structures in the semantics and logics of reactive systems. In Brauer, W., editor, *ICALP*, volume 194 of *Lecture Notes in Computer Science*, pages 15–32. Springer. 15

[Pratt, 1976] Pratt, V. R. (1976). Semantical considerations on Floyd-Hoare logic. In *FOCS*, pages 109–121. IEEE. 29

[Pratt, 1978] Pratt, V. R. (1978). A practical decision method for propositional dynamic logic: Preliminary report. In *STOC*, pages 326–337. ACM. 30

[Pratt, 1980] Pratt, V. R. (1980). A near-optimal method for reasoning about action. *J. Comput. Syst. Sci.*, 20(2):231–254. 30

[Pratt-Hartmann, 2005] Pratt-Hartmann, I. (2005). Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395. 26

[Presburger, 1929] Presburger, M. (1929). Über die Vollstandigkeit eines gewissen Systems der Arithmetik. In *Comptes rendus du I Congrés des Pays Slaves*. 24

[Queille and Sifakis, 1982] Queille, J.-P. and Sifakis, J. (1982). Specification and verification of concurrent systems in CESAR. In Dezani-Ciancaglini, M. and Montanari, U., editors, *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer. 15

[Rabin, 1969] Rabin, M. O. (1969). Decidability of second order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35. 27

[Reynolds, 2000] Reynolds, M. (2000). More past glories. In *LICS*, pages 229–240. 29

[Rosser, 1936] Rosser, J. B. (1936). Extensions of some theorems of Gödel and Church. *The Journal of Symbolic Logic*, 1:87–91. 24

[Schneider, 2004] Schneider, K. (2004). *Verification of Reactive Systems: Formal Methods and Algorithms*. SpringerVerlag. 47

[Seidl et al., 2003] Seidl, H., Schwentick, T., and Muscholl, A. (2003). Numerical document queries. In *PODS*, pages 155–166. ACM. 28

[Seidl et al., 2004] Seidl, H., Schwentick, T., Muscholl, A., and Habermehl, P. (2004). Counting in trees for free. In Díaz, J., Karhumäki, J., Lepistö, A., and Sannella, D., editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer. 24, 34, 35, 93, 94

[Stockmeyer, 1974] Stockmeyer, L. J. (1974). The complexity of decision problems in automata theory and logic. Technical Report MAC-TR-133, MIT. 28

[Tanabe et al., 2005] Tanabe, Y., Takahashi, K., Yamamoto, M., Tozawa, A., and Hagiya, M. (2005). A decision procedure for the alternation-free two-way modal μ-calculus. In Beckert, B., editor, *TABLEAUX*, volume 3702 of *Lecture Notes in Computer Science*, pages 277–291. Springer. 21, 31, 95

[Tarski, 1955] Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5(2):285–309. 44, 64

[ten Cate and Marx, 2009] ten Cate, B. and Marx, M. (2009). Axiomatizing the logical core of XPath 2.0. *Theory Comput. Syst.*, 44(4):561–589. 20, 38

[Thatcher and Wright, 1968] Thatcher, J. W. and Wright, J. B. (1968). Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81. 27

[Tobies, 1999] Tobies, S. (1999). A PSpace algorithm for graded modal logic. In Ganzinger, H., editor, *CADE*, volume 1632 of *Lecture Notes in Computer Science*, pages 52–66. Springer. 32

[Trakhtenbrot, 1957] Trakhtenbrot, B. A. (1957). On operators realizable in logical nets. *Dokl. Akad. Naut. SSSR*, 112:1005–1007. 15, 26, 27

[Turing, 1936] Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265. 24

[van Benthem, 1976] van Benthem, J. (1976). *Modal Correspondance Theory*. PhD thesis, University of Amsterdam. 28

[Vardi, 1998] Vardi, M. Y. (1998). Reasoning about the past with two-way automata. In Larsen, K. G., Skyum, S., and Winskel, G., editors, *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer. 30, 33

[Zee et al., 2008] Zee, K., Kuncak, V., and Rinard, M. C. (2008). Full functional verification of linked data structures. In Gupta, R. and Amarasinghe, S. P., editors, *PLDI*, pages 349–361. ACM. 24