Mu-Calculus Based Resolution of XPath Decision Problems

Pierre Genevès, Nabil Layaïda

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Mu-Calculus Based Resolution of XPath Decision Problems

Pierre Genevès — Nabil Layaïda

WAM, INRIA Rhône-Alpes

N° 5868

Mars 2006

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--5868--FR+ENG

Mu-Calculus Based Resolution of XPath Decision Problems

Pierre Genevès , Nabil Layaïda
WAM, INRIA Rhône-Alpes

Thème SYM — Systèmes symboliques
Projet WAM

Rapport de recherche n° 5868 — Mars 2006 — 30 pages

Abstract: XPath is the standard declarative notation for navigating XML data and returning a set of matching nodes. In the context of XSLT/XQuery analysis, query optimization, and XML type checking, XPath decision problems arise naturally. They notably include XPath containment (whether or not for any tree the result of a particular query is included in the result of a second one), and XPath satisfiability (whether or not an expression yields a non-empty result), in the presence (or the absence) of XML DTDs.

In this paper, we propose a unifying logic for XML, namely the alternation-free modal mu-calculus with converse. We show how to translate major XML concepts such as XPath and DTDs into this logic. Based on these embeddings, we show how XPath decision problems can be solved using a state-of-the-art EXPTIME decision procedure for mu-calculus satisfiability. We provide preliminary experiments which shed light, for the first time, on the cost of solving XPath decision problems in practice.

Key-words: XML, XPath, queries, types, containment, satisfiability, overlap, coverage, logic, mu-calculus

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Résolution de problèmes de décision XPath basée sur le mu-calcul

Résumé : XPath est le langage standard pour naviguer dans des données XML et renvoyer un ensemble de noeuds. Dans le contexte de l'analyse de XSLT/XQuery, de l'optimisation de requêtes, et de la vérification de types XML, les problèmes de décision de XPath surgissent naturellement. Ils incluent notamment l'inclusion de XPath (si pour n'importe quel arbre le résultat d'une requête est inclus dans le résultat d'une seconde), et la satisfaisabilité de XPath (si une expression donne un résultat non vide), en présence (ou absence) de DTDs XML. Dans cet article, nous proposons une logique unificatrice pour XML, à savoir le mu-calcul modal sans alternance avec programmes inverses. Nous montrons comment traduire les concepts principaux de XML tels que XPath et les DTDs dans cette logique. Grâce à ces traductions, nous montrons comment les problèmes de décision de XPath peuvent être résolus en utilisant une procédure de décision en temps exponentiel pour la satisfaisabilité du mu-calcul considéré. Nous menons des expérimentations qui permettent d'obtenir, pour la première fois, une évaluation empirique de la résolution des problèmes de décision de XPath.

Mots-clés : XML, XPath, requêtes, types, inclusion, intersection, satisfaisabilité, logique, mu-calcul

1 Introduction

XPath [9] is the standard declarative language for querying an XML tree and returning a set of nodes. It is increasingly popular due to its expressive power and its compact syntax. These advantages have given XPath a central role both in other key XML specifications and XML applications. It is used in XQuery as a core query language; in XSLT as node selector in the transformations; in XML Schema to define keys; in XLink and XPointer to reference portions of XML data. XPath is also used in many applications such as update languages [34] and XML access control [14].

Several XPath decision problems arise naturally in these use cases. The most basic decision problem for a query language is satisfiability [6]: whether or not an expression yields a non-empty result. XPath satisfiability is important for optimization of host languages implementations: for instance, if one can decide at compile time that a query is not satisfiable then subsequent bound computations can be avoided. Another basic decision problem is the XPath equivalence problem: whether or not two queries always return the same result. It is important for reformulation and optimization of the query itself, which aim at enforcing operational properties while preserving semantic equivalence [1, 26].

These two decision problems are reducible to XPath containment: whether or not, for any tree, the result of a particular query is included in the result of another one. Query containment is itself critical for static analysis of XML specifications and especially for type-checking transformations [27, 37].

Other XPath decision problems needed in applications include for example coverage (whether or not the nodes selected by an expression are always contained in the union of the results selected by several other expressions) and overlap (whether or not the intersection of the results of two expressions is empty).

A variety of factors contribute to the complexity of XPath decision problems such as the operators allowed in XPath queries and the combination of them. We present here the common distinctions between XPath fragments found in the literature, taken from [6]:

- positive vs. non-positive: depending whether the negation operator is considered or not inside qualifiers.
- downward vs. upward: depending whether queries specify downward or upward traversal of the tree, or both.
- recursive vs. non-recursive: depending whether XPath transitive closure axes (for instance “descendant” or “ancestor”) are considered or not.
- qualified vs. non-qualified: depending whether queries allow filtering predicates or not.
- with vs. without data values: depending whether comparisons of data values expressing joins are allowed or not.

$$
\begin{array}{llll}
\mathcal{L}_{\text{XPath}} & e & ::= & /p \mid p \mid e_1 \setminus e_2 \mid e_1 \cap e_2 \\
\mathit{Path} & p & ::= & p_1/p_2 \mid p[q] \mid a{::}n \\
\mathit{Qualifier} & q & ::= & q \text{ and } q \mid q \text{ or } q \mid \text{ not } q \mid p \\
\mathit{Axis} & a & ::= & \text{child} \mid \text{descendant} \mid \text{self} \mid \text{parent} \mid \text{ancestor} \mid \text{following} \mid \\
 & & & \text{preceding} \mid \text{descendant-or-self} \mid \text{ancestor-or-self} \mid \\
 & & & \text{preceding-sibling} \mid \text{following-sibling} \\
\mathit{NodeTest} & n & ::= & \sigma \mid *
\end{array}
$$

Figure 1: XPath Abstract Syntax.

From the results of [6, 33], we know that the combination of some previous factors with data values may lead to undecidability of decision problems such as the containment. In the remaining part of the paper, we focus on a large XPath fragment covering all factors except data values. This fragment (whose abstract syntax is given on Figure 1) is the largest considered so far in the literature.

XPath decision problems are also considered in the presence of XML types such as DTDs [8] or XML Schemas [13]. Combining XPath decision problems with types raises additional challenging research problems such as backward navigation in XML types.

In this paper, we are interested in finding an appropriate logic for XML, expressive enough to capture a significant range of XML decision problems, yet reasonably efficient in order to provide effective decision procedures.

From [28], we know that XPath expressive power is close to first-order logic (FO). However, FO does not fully capture regular tree types [7]. One of the most expressive (yet decidable) known logic is Monadic Second Order Logic (MSO) over tree structures, which extends FO by quantification over sets of nodes. Specifically, the appropriate MSO variant which exactly captures regular tree types is the weak monadic second-order logic of two successors (WS2S) [36, 12]. From [4, 25], we know that WS2S is exactly as expressive as the alternation-free fragment (AFMC) of the propositional modal μ-calculus introduced in [23]. However, the satisfiability problem for WS2S is non-elementary[1] while in EXPTIME[2] for AFMC. Moreover, the AFMC subsumes all early logics such as CTL [10] and PDL [15]. Furthermore, the work in [38] adds backward modalities to the propositional modal μ-calculus and shows that the resulting logic still admits an EXPTIME decision procedure for satisfiability.

[1] We recall that the term *elementary* introduced by Grzegorczyk [20] refers to functions obtained from some basic functions by operations of limited summation and limited multiplication. Consider the function $tower()$ defined by:

$$\begin{cases} tower(n,0) = n \\ tower(n,k+1) = 2^{tower(n,k)} \end{cases}$$

Grzegorczyk has shown that every elementary function in one argument is bounded by $\lambda n.tower(n,c)$ for some constant c. Hence, the term *non-elementary* refers to a function that grows faster than any such function.

[2] The complexity class EXPTIME is the set of all decision problems solvable by a deterministic Turing machine in $O(2^{p(n)})$ time, where $p(n)$ is a polynomial function of the input size n.

It follows that the alternation-free modal μ-calculus with backward modalities sounds as the ultimate logic for XML: expressive enough to capture a significant class of XPath decision problems, while potentially providing efficient and practically effective decision procedures.

In this paper, we propose the alternation free modal μ-calculus with backward modalities as the appropriate logic for effectively solving XPath decision problems. We present a linear translation of a large XPath fragment into μ-calculus. In addition, we embed regular tree types, including DTDs, in the μ-calculus. This yields decision procedures for XPath decision problems needed in applications such as XPath satisfiability, containment, equivalence, overlap, coverage, in the absence or in the presence of DTDs. We build on our translations of XML concepts and show how to effectively answer XML decision problems in practice. Using a state-of-the-art EXPTIME decision procedure for μ-calculus satisfiability, we give preliminary experimental results. These results shed light, for the first time, on the cost of solving XML decision problems in practice.

The remaining part of the paper is organized as follows: in Section 2 we introduce the logic we propose for reasoning on XML trees; in Section 3 we describe the translation of XPath queries into this logic; Section 4 embeds regular XML types into the logic. Based on these translations, Section 5 explains how to formulate and solve common decision problems. We present experimental results in Section 6, before discussing related work in Section 7 and concluding in Section 8.

2 A Logic for XML

We consider an XML document as a finite ordered and labeled tree of unbounded depth and arity. Tree nodes are labeled with symbols taken from a countably infinite set Σ. There is a straightforward isomorphism between sequences of unranked trees and binary trees. In order to describe it, we first define the set $\mathcal{T}_\Sigma^n$ of unranked trees:

$$\mathcal{T}_\Sigma^n \ni t ::= \sigma(h)$$

where $\sigma \in \Sigma$ and h is a hedge, i.e. a sequence of unranked trees, defined as follows:

$$\mathcal{H}_\Sigma \ni h ::= \sigma(h), h' \mid ()$$

A binary tree t is either a σ-labeled root of two subtrees ($\sigma \in \Sigma$) or the empty tree:

$$\mathcal{T}_\Sigma^2 \ni t ::= \sigma(t, t') \mid \epsilon$$

Unranked trees can be translated into binary trees with the following function:

$$\begin{array}{lcl} \beta(\cdot) & : & \mathcal{H}_\Sigma \to \mathcal{T}_\Sigma^2 \\ \beta(\sigma(h), h') & = & \sigma(\beta(h), \beta(h')) \\ \beta(()) & = & \epsilon \end{array}$$

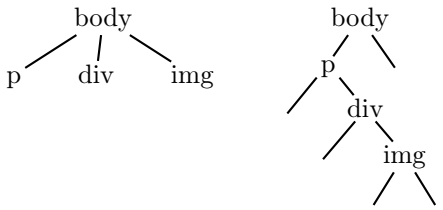

Figure 2: N-ary and Binary Tree Representations.

The inverse translation function converts a binary tree into a sequence of unranked trees:

$$\begin{array}{lcl} \beta^{-1}(\cdot) & : & \mathcal{T}_\Sigma^2 \rightarrow \mathcal{H}_\Sigma \\ \beta^{-1}(\sigma(t,t')) & = & \sigma(\beta^{-1}(t)), \beta^{-1}(t') \\ \beta^{-1}(\epsilon) & = & () \end{array}$$

For example, Figure 2 illustrates how a sample unranked tree is mapped to its binary representation and vice-versa.

Note that the translation of a single unranked tree results in a binary tree of the form $\sigma(t, \epsilon)$. Reciprocally, the inverse translation of such a binary tree always yields a single unranked tree. When modeling XML, we therefore restrict our attention to binary trees of the form $\sigma(t, \epsilon)$, without loss of generality.

We now introduce the logic we propose for reasoning over these structures.

2.1 The μ-Calculus

The *propositional μ-calculus* is a propositional modal logic extended with least and greatest fixpoint operators [23]. A *signature* Ξ for the μ-calculus consists of a set *Prop* of atomic propositions, a set *Var* of propositional variables, and a set *Prog* of atomic programs. In the XML context, atomic propositions represent the symbols of the alphabet Σ used to label XML trees. Atomic programs allow navigation in trees.

The μ-calculus with backward modalities[3] [38] augments the propositional μ-calculus by associating with each atomic program a its converse $\overline{a}$. A *program* α is either an atomic program or its converse. This is the only difference between the propositional μ-calculus that lacks backward modalities. It is important to note that the addition of converse programs preserve the EXPTIME upper bound for the satisfiability problem [38].

The set $\mathcal{L}_\mu$ of formulae of the μ-calculus with backward modalities over the signature Ξ is defined as follows:

$$\begin{array}{rl} \mathcal{L}_\mu \ni \varphi ::= & \top \mid \bot \mid p \mid \neg\varphi \mid \varphi_1 \ \vee \ \varphi_2 \mid \varphi_1 \ \wedge \ \varphi_2 \mid \\ & [\alpha]\,\varphi \mid \ \langle\alpha\rangle\,\varphi \mid X \mid \mu X.\varphi \mid \nu X.\varphi \end{array}$$

[3]The μ-calculus with backward modalities is also known as the *full* μ-calculus, or alternatively as the *two-way* μ-calculus in the literature.

$$
\begin{array}{lcl}
[\![\cdot]\!]^K_V & : & \mathcal{L}_\mu \longrightarrow 2^W \\
[\![\top]\!]^K_V & = & W \\
[\![\bot]\!]^K_V & = & \emptyset \\
[\![p]\!]^K_V & = & L(p) \\
[\![\neg\varphi]\!]^K_V & = & W \setminus [\![\varphi]\!]^K_V \\
[\![\varphi_1 \vee \varphi_2]\!]^K_V & = & [\![\varphi_1]\!]^K_V \cup [\![\varphi_2]\!]^K_V \\
[\![\varphi_1 \wedge \varphi_2]\!]^K_V & = & [\![\varphi_1]\!]^K_V \cap [\![\varphi_2]\!]^K_V \\
[\![[\alpha]\,\varphi]\!]^K_V & = & \{w : \forall w'(w,w') \in R(\alpha) \Rightarrow w' \in [\![\varphi]\!]^K_V\} \\
[\![\langle\alpha\rangle\,\varphi]\!]^K_V & = & \{w : \exists w'(w,w') \in R(\alpha) \wedge w' \in [\![\varphi]\!]^K_V\} \\
[\![\mu X.\varphi]\!]^K_V & = & \bigcap\{W' \subseteq W : [\![\varphi]\!]^K_{V[X/W']} \subseteq W'\} \\
[\![\nu X.\varphi]\!]^K_V & = & \bigcup\{W' \subseteq W : [\![\varphi]\!]^K_{V[X/W']} \supseteq W'\}
\end{array}
$$

Figure 3: Semantics of the μ-Calculus.

where $p \in \mathit{Prop}$, $X \in \mathit{Var}$ and α is a program.

The semantics of the full μ-calculus is given with respect to a *Kripke structure* $K = \langle W, R, L\rangle$ where W is a set of nodes, $R : \mathit{Prog} \to 2^{W\times W}$ assigns to each atomic program a transition relation over W, and L is an interpretation function that assigns to each atomic proposition a set of nodes.

The formal semantics function $[\![\cdot]\!]^K_V$ shown on Figure 3 defines the semantics of a μ-calculus formula in terms of a Kripke structure K and a valuation V. A valuation $V : \mathit{Var} \to 2^W$ maps each variable to a subset of W. For a valuation V, a variable X, and a set of nodes $W' \subseteq W$, $V[X/W']$ denotes the valuation that is obtained from V by assigning W' to X.

Note that if φ is a sentence (i.e. all propositional variables occurring in φ are bound), then no valuation is required. For a node $w \in W$ and a sentence φ, we say that φ holds at w in K, denoted $K, w \models \varphi$ iff $w \in [\![\varphi]\!]^K$.

The two modalities $\langle a\rangle\,\varphi$ (possibility) and $[a]\,\varphi$ (necessity) are operators for navigating the structure.

The syntax of $\mathcal{L}_\mu$ formulae as given previously is in fact redundant. Actually, we only have to deal with a subset of $\mathcal{L}_\mu$ composed of formulae in negation normal form. We say that a formula is in *negation normal form* if and only if all negations in the formula appear only before atomic propositions. Every formula is equivalent to a formula in negation normal form [23], which can be obtained by expanding negations using De Morgan's rules together with standard dualities for modalities and fixpoints (c.f. Figure 4). For readability purposes, translations of XPath expressions given in Section 3 are not given in negation normal form.

For XML, we are in fact interested by a specific subset of $\mathcal{L}_\mu$, namely the alternation-free modal-μ-calculus with backward modalities over finite binary trees.

$$
\begin{array}{lcl}
\neg\,[\alpha]\,\varphi & = & \langle\alpha\rangle\,\neg\varphi \\
\neg\,\langle\alpha\rangle\,\varphi & = & [\alpha]\,\neg\varphi \\
\neg\mu X.\varphi & = & \nu X.\neg\varphi[X/\neg X] \\
\neg\nu X.\varphi & = & \mu X.\neg\varphi[X/\neg X] \\
\neg(\varphi_1 \wedge \varphi_2) & = & \neg\varphi_1 \vee \neg\varphi_2 \\
\neg(\varphi_1 \vee \varphi_2) & = & \neg\varphi_1 \wedge \neg\varphi_2 \\
\neg\neg\varphi & = & \varphi
\end{array}
$$

Figure 4: Dualities for Negation Normal Form.

We recall that a $\mathcal{L}_\mu$ formula φ in negation normal form is *alternation-free* whenever the following condition holds[4]: if $\mu X.\varphi_1$ (respectively $\nu X.\varphi_1$) is a subformula of φ and $\nu Y.\varphi_2$ (respectively $\mu Y.\varphi_2$) is a subformula of φ_1 then X does not occur freely in φ_2.

The following section now introduces the additional restrictions of $\mathcal{L}_\mu$ related to finite binary trees.

2.2 XML Constraints on Kripke Structures

In this section, we restrict the satisfiability problem of $\mathcal{L}_\mu$ over Kripke structures to the satisfiability problem over finite binary trees.

The propositional μ-calculus has the *finite tree model property*: a formula that is satisfiable, is also satisfiable on a finite tree [24]. Unfortunately, the introduction of backward modalities causes the loss of the finite model property [38]. Therefore, we need to reinforce the finite model property and introduce some others to ensure we work on finite binary trees encoding XML structures.

First, each XML node has at most one Σ-label, i.e. $p \wedge p'$ never holds for distinct atomic propositions p and p'.

Second, for navigating binary trees, we only use two atomic programs 1 and 2, and their associated relations $R(1) = \prec_{\text{fc}}$ and $R(2) = \prec_{\text{ns}}$ whose meaning is to respectively connect a node to its left child and to its right child. For any $(x, y) \in W \times W$, $x \prec_{\text{fc}} y$ holds iff y is the left child of x (i.e. the first child in the unranked tree representation) and $x \prec_{\text{ns}} y$ holds iff y is the right child of x in the binary tree representation (i.e. the next sibling in the unranked tree representation).

For each atomic program $a \in \{1, 2\}$ we define $R(\overline{a})$ to be the relational inverse of $R(a)$, i.e., $R(\overline{a}) = \{(v, u) : (u, v) \in R(a)\}$. We thus consider programs $\alpha \in \{1, 2, \overline{1}, \overline{2}\}$ inside modalities for navigating forward and backward.

We now define restrictions for a Kripke structure to form a finite binary tree [35]. A Kripke structure $t = \langle W, R, L\rangle$ is a finite binary tree if it satisfies the following conditions:

(1) W is finite

[4]For instance, $\nu X.(\mu Y.\,\langle 1\rangle\, Y \wedge p) \vee \langle 2\rangle\, X$ is alternation-free but $\nu X.(\mu Y.\,\langle 1\rangle\, Y \wedge X) \vee p$ is not since X bound by ν appears freely in the scope of μY.

(2) the set of nodes W together with the accessibility relation $\prec_{\mathrm{fc}} \cup \prec_{\mathrm{ns}}$ define a tree

(3) $\prec_{\mathrm{fc}}$ and $\prec_{\mathrm{ns}}$ are partial functions, i.e. for all $m \in W$ and $j \in \{1, 2\}$ there is at most one $m_j \in W$ such that $(m, m_j) \in R(j)$.

We say that a finite binary tree $t = \langle W, R, L\rangle$ satisfies φ if $t, r \models \varphi$ where $r \in W$ is the root of the tree t.

For accessing the root, we use the $\mathcal{L}_\mu$ formula

$$\varphi_{\mathrm{root}} = \left[\overline{1}\right] \bot \wedge \left[\overline{2}\right] \bot$$

which selects a node provided it has no parent.

For ensuring finiteness, we rely on König's lemma which states that *a finitely branching infinite tree has some infinite path* or, in other words, a finitely branching tree in which every branch is finite is finite. The expression $\nu X.\, \langle 1\rangle\, X \vee \langle 2\rangle\, X$ is only satisfied by structures containing infinite or cyclic paths. To prevent the existence of such paths, we negate the previous formula and, by propagating negation using the rules presented on Figure 4, we obtain:

$$\varphi_{\mathrm{ft}} = \mu X.\, [1]\, X \wedge [2]\, X$$

φ_{ft} states that all descending branches are finite from the current context node. In our case we need φ_{ft} to hold at the root (i.e. $\varphi_{\mathrm{root}} \wedge \varphi_{\mathrm{ft}}$ must hold), in order to ensure we work with a finite structure. This is for condition (1) to be satisfied.

We still need to enforce (2) and (3). We do this by rewriting existential modalities in such a way that if a successor is supposed to exist, then there exists at least one, and if there are many all verify the same property. This is a way to overcome the difficulty that in μ-calculus, one cannot naturally express a property like "a node has exactly n successors". Technically, we denote by φ^{FBT} the formula φ where all occurrences of $\langle\alpha\rangle\, \psi$ are replaced by $\langle\alpha\rangle\, \top \wedge [\alpha]\, \psi^{\mathrm{FBT}}$. This replacement is enough to enforce conditions (2) and (3).

Proposition 1 *A $\mathcal{L}_\mu$ formula φ is satisfied by a finite binary tree model if and only if the formula $\varphi_{root} \wedge \varphi_{ft} \wedge \varphi^{FBT}$ is satisfied by a Kripke structure.*

The detailed proof is described in [35]. The "if" part iteratively constructs a tree model and proceeds by induction on the structure on φ. The "only if" part is almost immediate.

Proposition 1 gives the adequate framework for formulating decision problems on XML structures in terms of a μ-calculus formula.

3 XPath

XPath expressions are directly used for querying unranked XML trees. We first recall XPath denotational semantics over unranked trees [39]. The evaluation of an XPath query returns

a set of nodes reachable from a context node x in a tree t. The formal semantics functions $\mathcal{S}_e$ and $\mathcal{S}_p$ define the set of nodes respectively returned by expressions and paths:

$$
\begin{array}{lcl}
\mathcal{S}_e[\![\cdot]\!]: & : & \mathcal{L}_{\text{XPath}} \times \mathit{Node} \times \mathcal{T}_\Sigma^n \longrightarrow \text{Set}(\mathit{Node}) \\
\mathcal{S}_e[\![/p]\!]_x^t & = & \mathcal{S}_p[\![p]\!]_{root()}^t \\
\mathcal{S}_e[\![p]\!]_x^t & = & \mathcal{S}_p[\![p]\!]_x^t \\
\mathcal{S}_e[\![e_1 \mid e_2]\!]_x^t & = & \mathcal{S}_e[\![e_1]\!]_x^t \cup \mathcal{S}_e[\![e_2]\!]_x^t \\
\mathcal{S}_e[\![e_1 \cap e_2]\!]_x^t & = & \mathcal{S}_e[\![e_1]\!]_x^t \cap \mathcal{S}_e[\![e_2]\!]_x^t
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{S}_p[\![\cdot]\!]: & : & \mathit{Path} \times \mathit{Node} \times \mathcal{T}_\Sigma^n \longrightarrow \text{Set}(\mathit{Node}) \\
\mathcal{S}_p[\![p_1/p_2]\!]_x^t & = & \{x_2 \mid x_1 \in \mathcal{S}_p[\![p_1]\!]_x^t \wedge x_2 \in \mathcal{S}_p[\![p_2]\!]_{x_1}^t\} \\
\mathcal{S}_p[\![p[q]]\!]_x^t & = & \{x_1 \mid x_1 \in \mathcal{S}_p[\![p]\!]_x^t \wedge \mathcal{S}_q[\![q]\!]_{x_1}^t\} \\
\mathcal{S}_p[\![\text{a::}\sigma]\!]_x^t & = & \{x_1 \mid x_1 \in \mathcal{S}_a[\![\text{a}]\!]_x^t \wedge \mathit{name}(x_1) = \sigma\} \\
\mathcal{S}_p[\![\text{a::}*]\!]_x^t & = & \{x_1 \mid x_1 \in \mathcal{S}_a[\![\text{a}]\!]_x^t\}
\end{array}
$$

The function $\mathcal{S}_q$ defines the semantics of qualifiers that basically state the existence (or absence) of one or more paths from a context node x:

$$
\begin{array}{lcl}
\mathcal{S}_q[\![\cdot]\!]: & : & \mathit{Qualifier} \times \mathit{Node} \times \mathcal{T}_\Sigma^n \longrightarrow \mathit{Boolean} \\
\mathcal{S}_q[\![q_1 \text{ and } q_2]\!]_x^t & = & \mathcal{S}_q[\![q_1]\!]_x^t \wedge \mathcal{S}_q[\![q_2]\!]_x^t \\
\mathcal{S}_q[\![q_1 \text{ or } q_2]\!]_x^t & = & \mathcal{S}_q[\![q_1]\!]_x^t \vee \mathcal{S}_q[\![q_2]\!]_x^t \\
\mathcal{S}_q[\![\text{not } q]\!]_x^t & = & \neg\, \mathcal{S}_q[\![q]\!]_x^t \\
\mathcal{S}_q[\![p]\!]_x^t & = & \mathcal{S}_p[\![p]\!]_x^t \neq \emptyset
\end{array}
$$

Eventually the function $\mathcal{S}_a$ gives the denotational semantics of axes:

$$
\begin{array}{lcl}
\mathcal{S}_a[\![\cdot]\!]: & : & \mathit{Axis} \times \mathit{Node} \times \mathcal{T}_\Sigma^n \longrightarrow \text{Set}(\mathit{Node}) \\
\mathcal{S}_a[\![\text{child}]\!]_x^t & = & \mathit{children}(x) \\
\mathcal{S}_a[\![\text{parent}]\!]_x^t & = & \mathit{parent}(x) \\
\mathcal{S}_a[\![\text{descendant}]\!]_x^t & = & \mathit{children}^+(x) \\
\mathcal{S}_a[\![\text{ancestor}]\!]_x^t & = & \mathit{parent}^+(x) \\
\mathcal{S}_a[\![\text{self}]\!]_x^t & = & \{x\} \\
\mathcal{S}_a[\![\text{descendant-or-self}]\!]_x^t & = & \mathcal{S}_a[\![\text{descendant}]\!]_x^t \cup \mathcal{S}_a[\![\text{self}]\!]_x^t \\
\mathcal{S}_a[\![\text{ancestor-or-self}]\!]_x^t & = & \mathcal{S}_a[\![\text{ancestor}]\!]_x^t \cup \mathcal{S}_a[\![\text{self}]\!]_x^t \\
\mathcal{S}_a[\![\text{preceding}]\!]_x^t & = & \{y \mid y \ll x\} \setminus \mathcal{S}_a[\![\text{ancestor}]\!]_x^t \\
\mathcal{S}_a[\![\text{following}]\!]_x^t & = & \{y \mid x \ll y\} \setminus \mathcal{S}_a[\![\text{descendant}]\!]_x^t \\
\mathcal{S}_a[\![\text{following-sibling}]\!]_x^t & = & \{y \mid y \in \mathit{child}(\mathit{parent}(x)) \wedge x \ll y\} \\
\mathcal{S}_a[\![\text{preceding-sibling}]\!]_x^t & = & \{y \mid y \in \mathit{child}(\mathit{parent}(x)) \wedge y \ll x\}
\end{array}
$$

in which $\mathit{root}()$, $\mathit{children}(x)$ and $\mathit{parent}(x)$ are primitives for navigating unranked trees, $\ll$ is the ordering relation ($x \ll y$ holds if and only if the node x is before the node y in the depth-first traversal order of the tree), and $\mathit{name}()$ is the mean to access the labeling of the tree.

3.1 A Translation into the μ-Calculus

We now explain how an XPath expression can be translated into an equivalent formula in $\mathcal{L}_\mu$ over binary trees. The translation adheres to XPath formal semantics given above in the sense that the translated formula holds for nodes which are selected by the XPath query. Navigation as performed by XPath in unranked trees is translated in terms of navigation in the binary tree representation.

3.1.1 Logical Interpretation of Axes

We first translate navigational primitives, namely XPath axes. The translation is formally specified on Figure 6 as a translation function noted "$A^{\rightarrow}[\![\cdot]\!](\cdot)$" which takes an XPath axis as input, and returns its translation in μ-calculus, in terms of the μ-calculus formula given as a parameter to allow further composition. $A^{\rightarrow}[\![a]\!](\chi)$ holds for all nodes that can be accessed through the axis a from some node verifying χ.

Figure 7 gives the intuition of the translation of the XPath axis "child". In this case, we start from a context, designated by the formula χ. Children of a node in the binary tree representation form the inductively defined set of nodes composed of the left child and closed under the $\prec_{\text{ns}}$ relation. Recursion in the right branch starting from the left child is captured by a least fixpoint.

Other axis translations are built in a similar manner. Note that since we want the translated formula to hold for target nodes which are selected by the axis, inverse modalities are involved.

For readers more familiar with PDL and CPDL (PDL with converse programs) both defined in [15], we give a correspondence of notations on Figure 5.

3.1.2 Logical Interpretation of Expressions

The translation of XPath expressions into μ-calculus is given on Figure 8. It is formally expressed as a translation function noted "$E^{\rightarrow}[\![\cdot]\!](\cdot)$" which takes an XPath expression as input, a μ-calculus formula as a parameter which indicates the context from which the expression is applied. Absolute XPath expressions are interpreted from the root (selected by the μ-calculus expression φ_{root}), whereas relative expressions are interpreted relatively to any context node.

The translation of expressions relies on the translations of paths shown on Figure 9. XPath most essential construct p_1/p_2 translates into formula composition in $\mathcal{L}_\mu$, such that the resulting formula holds for all nodes accessed through p_2 from those nodes accessed from χ by p_1.

The translation of the branching construct $p[q]$ significantly differs. The resulting formula must hold for all nodes that can be accessed through p and from which q holds (c.f. XPath denotational semantics given in Section 3). To preserve semantics, the translation of $p[q]$ stops the "selecting navigation" to those nodes reached by p, then filters them depending whether q holds or not. We express this by introducing a dual formal translation function

XPath	μ-Calculus
$\pi/\text{following-sibling::}*$	$\mu Z.\, \langle\overline{2}\rangle\, \pi \vee \langle\overline{2}\rangle\, Z$
$\pi/\text{child::}*$	$\mu Z.\, \langle\overline{1}\rangle\, \pi \vee \langle\overline{2}\rangle\, Z$
$\pi/\text{descendant::}*$	$\mu Z.\, \langle\overline{1}\rangle\, (\pi \vee Z) \vee \langle\overline{2}\rangle\, Z$
$\pi/\text{descendant-or-self::}*$	$\mu Z.\pi \vee \mu Y.\, \langle\overline{1}\rangle\, (Y \vee Z) \vee \langle\overline{2}\rangle\, Y$
$\pi/\text{parent::}*$	$\langle 1\rangle\, \mu Z.\pi \vee \langle 2\rangle\, Z$
$\pi/\text{ancestor::}*$	$\langle 1\rangle\, \mu Z.\pi \vee \langle 1\rangle\, Z \vee \langle 2\rangle\, Z$
$\pi/\text{ancestor-or-self::}*$	$\mu Z.\pi \vee \mu Y.\, \langle 1\rangle\, (Y \vee Z) \vee \langle 2\rangle\, Y$
$\pi/\text{preceding-sibling::}*$	$\mu Z.\, \langle 2\rangle\, \pi \vee \langle 2\rangle\, Z$

XPath	CPDL
$\pi/\text{following-sibling::}*$	$\left\langle \overline{2}^* \cdot \overline{2} \right\rangle \pi$
$\pi/\text{child::}*$	$\left\langle \overline{2}^* \cdot \overline{1} \right\rangle \pi$
$\pi/\text{descendant::}*$	$\langle (\overline{1}\vert\overline{2})^* \cdot \overline{1} \rangle\, \pi$
$\pi/\text{descendant-or-self::}*$	$\langle \text{nil}\vert(\overline{1}\vert\overline{2})^* \cdot \overline{1} \rangle\, \pi$
$\pi/\text{parent::}*$	$\langle 1 \cdot 2^* \rangle\, \pi$
$\pi/\text{ancestor::}*$	$\langle 1 \cdot (1\vert 2)^* \rangle\, \pi$
$\pi/\text{ancestor-or-self::}*$	$\langle \text{nil}\vert 1 \cdot (1\vert 2)^* \rangle\, \pi$
$\pi/\text{preceding-sibling::}*$	$\langle 2^* \cdot 2 \rangle\, \pi$

Figure 5: Logical Correspondences in terms of the Early CPDL Operators.

$A^{\rightarrow}[\![\cdot]\!](\cdot)$	:	$Axis \times \mathcal{L}_\mu \longrightarrow \mathcal{L}_\mu$
$A^{\rightarrow}[\![\text{self}]\!](\chi)$	=	χ
$A^{\rightarrow}[\![\text{following-sibling}]\!](\chi)$	=	$\mu Z.\, \langle\overline{2}\rangle\, \chi \vee \langle\overline{2}\rangle\, Z$
$A^{\rightarrow}[\![\text{child}]\!](\chi)$	=	$\mu Z.\, \langle\overline{1}\rangle\, \chi \vee \langle\overline{2}\rangle\, Z$
$A^{\rightarrow}[\![\text{descendant}]\!](\chi)$	=	$\mu Z.\, \langle\overline{1}\rangle\, (\chi \vee Z) \vee \langle\overline{2}\rangle\, Z$
$A^{\rightarrow}[\![\text{descendant-or-self}]\!](\chi)$	=	$\mu Z.\chi \vee \mu Y.\, \langle\overline{1}\rangle\, (Y \vee Z) \vee \langle\overline{2}\rangle\, Y$
$A^{\rightarrow}[\![\text{parent}]\!](\chi)$	=	$\langle 1\rangle\, \mu Z.\chi \vee \langle 2\rangle\, Z$
$A^{\rightarrow}[\![\text{ancestor}]\!](\chi)$	=	$\langle 1\rangle\, \mu Z.\chi \vee \langle 1\rangle\, Z \vee \langle 2\rangle\, Z$
$A^{\rightarrow}[\![\text{ancestor-or-self}]\!](\chi)$	=	$\mu Z.\chi \vee \mu Y.\, \langle 1\rangle\, (Y \vee Z) \vee \langle 2\rangle\, Y$
$A^{\rightarrow}[\![\text{preceding-sibling}]\!](\chi)$	=	$\mu Z.\, \langle 2\rangle\, \chi \vee \langle 2\rangle\, Z$
$A^{\rightarrow}[\![\text{following}]\!](\chi)$	=	$A^{\rightarrow}[\![\text{descendant-or-self}]\!](A^{\rightarrow}[\![\text{following-sibling}]\!](\eta))$
$A^{\rightarrow}[\![\text{preceding}]\!](\chi)$	=	$A^{\rightarrow}[\![\text{descendant-or-self}]\!](A^{\rightarrow}[\![\text{preceding-sibling}]\!](\eta))$
		where η is a shorthand for $A^{\rightarrow}[\![\text{ancestor-or-self}]\!](\chi)$

Figure 6: Translation of XPath Axes.

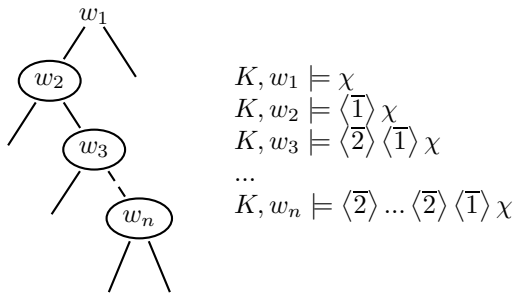

Figure 7: Intuition for the "child" Axis.

$$
\begin{array}{lcl}
E^{\rightarrow}[\![\cdot]\!](\cdot) & : & \mathcal{L}_{\mathrm{XPath}} \times \mathcal{L}_{\mu} \longrightarrow \mathcal{L}_{\mu} \\
E^{\rightarrow}[\![/p]\!](\chi) & = & P^{\rightarrow}[\![p]\!]([\overline{1}]\, \bot \wedge [\overline{2}]\, \bot) \\
E^{\rightarrow}[\![p]\!](\chi) & = & P^{\rightarrow}[\![p]\!](\chi) \\
E^{\rightarrow}[\![e_1 \cup e_2]\!](\chi) & = & E^{\rightarrow}[\![e_1]\!](\chi) \vee E^{\rightarrow}[\![e_2]\!](\chi) \\
E^{\rightarrow}[\![e_1 \cap e_2]\!](\chi) & = & E^{\rightarrow}[\![e_1]\!](\chi) \wedge E^{\rightarrow}[\![e_2]\!](\chi)
\end{array}
$$

Figure 8: Translation of Expressions.

$$
\begin{array}{lcl}
P^{\rightarrow}[\![\cdot]\!](\cdot) & : & \mathit{Path} \times \mathcal{L}_{\mu} \longrightarrow \mathcal{L}_{\mu} \\
P^{\rightarrow}[\![p_1/p_2]\!](\chi) & = & P^{\rightarrow}[\![p_2]\!](P^{\rightarrow}[\![p_1]\!](\chi)) \\
P^{\rightarrow}[\![p[q]]\!](\chi) & = & P^{\rightarrow}[\![p]\!](\chi) \wedge Q^{\leftarrow}[\![q]\!](\top) \\
P^{\rightarrow}[\![a{::}\sigma]\!](\chi) & = & A^{\rightarrow}[\![a]\!](\chi) \wedge \sigma \\
P^{\rightarrow}[\![a{::}*]\!](\chi) & = & A^{\rightarrow}[\![a]\!](\chi)
\end{array}
$$

Figure 9: Translation of Paths.

$$
\begin{array}{lcl}
Q^{\leftarrow}[\![\cdot]\!](\cdot) & : & \mathit{Qualifier} \times \mathcal{L}_\mu \longrightarrow \mathcal{L}_\mu \\
Q^{\leftarrow}[\![q_1 \text{ and } q_2]\!](\chi) & = & Q^{\leftarrow}[\![q_1]\!](\chi) \wedge Q^{\leftarrow}[\![q_2]\!](\chi) \\
Q^{\leftarrow}[\![q_1 \text{ or } q_2]\!](\chi) & = & Q^{\leftarrow}[\![q_1]\!](\chi) \vee Q^{\leftarrow}[\![q_2]\!](\chi) \\
Q^{\leftarrow}[\![\text{not} q]\!](\chi) & = & \neg\, Q^{\leftarrow}[\![q]\!](\chi) \\
Q^{\leftarrow}[\![p]\!](\chi) & = & P^{\leftarrow}[\![p]\!](\chi) \\
 & & \\
P^{\leftarrow}[\![\cdot]\!](\cdot) & : & \mathit{Path} \times \mathcal{L}_\mu \longrightarrow \mathcal{L}_\mu \\
P^{\leftarrow}[\![p_1/p_2]\!](\chi) & = & P^{\leftarrow}[\![p_1]\!](P^{\leftarrow}[\![p_2]\!](\chi)) \\
P^{\leftarrow}[\![p[q]]\!](\chi) & = & P^{\leftarrow}[\![p]\!](\chi \wedge Q^{\leftarrow}[\![q]\!](\top)) \\
P^{\leftarrow}[\![a{::}\sigma]\!](\chi) & = & A^{\leftarrow}[\![a]\!](\chi \wedge \sigma) \\
P^{\leftarrow}[\![a{::}*]\!](\chi) & = & A^{\leftarrow}[\![a]\!](\chi)
\end{array}
$$

Figure 10: Translation of Qualifiers.

for XPath qualifiers, noted $Q^{\leftarrow}[\![\cdot]\!](\cdot)$ (and shown on Figure 10), which performs "filtering" instead of navigation. Specifically, $P^{\rightarrow}[\![\cdot]\!](\cdot)$ can be seen as the "navigational" translating function: the translated formula holds for target nodes of the given path. On the opposite, $Q^{\leftarrow}[\![\cdot]\!](\cdot)$ can be seen as the "filtering" translating function: it states the existence of a path without moving to its result. The translated formula $Q^{\leftarrow}[\![q]\!](\chi)$ (respectively $P^{\leftarrow}[\![p]\!](\chi)$) holds for nodes from which there exists a qualifier q (respectively a path p) leading to a node verifying χ.

XPath translation into μ-calculus is based on these two translating "modes", the first one being used for paths and the second one for qualifiers. Note that whenever the "filtering" mode is entered, it will never be leaved. This differs from the denotational semantics given in Section 3 in which the formal semantics functions for paths and qualifiers are mutually recursive (and cause naive implementations to be unnecessarily complex, as pointed by [17]). Translations of paths inside qualifiers are also given on Figure 10. They use the specific translations for axes inside qualifiers, based on XPath symmetry, shown on Figure 11.

The cost of the translation is linear in the length of the XPath expression since there is no duplication of subformulae of arbitrary length in the formal translations. Formulae in which the formal parameter χ appears twice (see Figure 8 and Figure 10) do not cause such duplication since the value of χ is either $\top$ or φ_{root} constants.

Note that the translation of an XPath expression is a sentence. Indeed, for absolute XPath expressions, the translation starts from the root (the initial formal parameter is φ_{root}). For relative expressions, the translated formula is closed by the initial formal parameter $\top$ (all nodes).

We can prove that the translated $\mathcal{L}_\mu$ formula over binary trees is semantically equivalent to the original XPath expression over corresponding unranked trees. For instance, if we relate our translations in $\mathcal{L}_\mu$ to the XPath denotational semantics given in Section 3:

$$
\begin{array}{lcl}
A^{\leftarrow}[\![\cdot]\!](\cdot) & : & \mathit{Axis} \times \mathcal{L}_\mu \longrightarrow \mathcal{L}_\mu \\
A^{\leftarrow}[\![\text{self}]\!](\chi) & = & \chi \\
A^{\leftarrow}[\![\text{following-sibling}]\!](\chi) & = & A^{\rightarrow}[\![\text{preceding-sibling}]\!](\chi) \\
A^{\leftarrow}[\![\text{child}]\!](\chi) & = & A^{\rightarrow}[\![\text{parent}]\!](\chi) \\
A^{\leftarrow}[\![\text{descendant}]\!](\chi) & = & A^{\rightarrow}[\![\text{ancestor}]\!](\chi) \\
A^{\leftarrow}[\![\text{descendant-or-self}]\!](\chi) & = & A^{\rightarrow}[\![\text{ancestor-or-self}]\!](\chi) \\
A^{\leftarrow}[\![\text{parent}]\!](\chi) & = & A^{\rightarrow}[\![\text{child}]\!](\chi) \\
A^{\leftarrow}[\![\text{ancestor}]\!](\chi) & = & A^{\rightarrow}[\![\text{descendant}]\!](\chi) \\
A^{\leftarrow}[\![\text{ancestor-or-self}]\!](\chi) & = & A^{\rightarrow}[\![\text{descendant-or-self}]\!](\chi) \\
A^{\leftarrow}[\![\text{preceding-sibling}]\!](\chi) & = & A^{\rightarrow}[\![\text{following-sibling}]\!](\chi) \\
A^{\leftarrow}[\![\text{following}]\!](\chi) & = & A^{\rightarrow}[\![\text{preceding}]\!](\chi) \\
A^{\leftarrow}[\![\text{preceding}]\!](\chi) & = & A^{\rightarrow}[\![\text{following}]\!](\chi)
\end{array}
$$

Figure 11: Symmetry of Axes inside Qualifiers.

Proposition 1 *Let t' be an XML tree, t its binary representation, x' and y' nodes in t', and y the image of y' in t, then for an XPath expression e:*

$$(\forall x', y' \in \mathcal{S}_e[\![e]\!]^{t'}_{x'}) \text{ iff } t, y \models \varphi_{root} \wedge \varphi_{ft} \wedge (E^{\rightarrow}[\![e]\!](\top))^{FBT}$$

The proof is done by a straightforward structural induction that "peels off" the compositional layers of each set of rules. This result links XPath decision problems in the absence of XML types to satisfiability in $\mathcal{L}_\mu$. We now show how XML types can also be translated in the μ-calculus.

4 XML Types

XML types describe structural constraints for XML documents. Several formalisms exist for describing classes of XML documents (see [31] for an overview). In this paper, we translate the class of regular tree languages, that gathers all widely used formalisms for describing types of XML documents (including the well-known DTDs) into $\mathcal{L}_\mu$ over binary trees.

We begin with the syntactic definition of tree type expressions. We define a type T as follows:

$$
\begin{array}{rcl}
\mathcal{L}_{CFT} \ni T & ::= & \emptyset \mid () \mid X \mid l[T] \mid T_1, T_2 \mid T_1 \shortmid T_2 \mid \\
& & \text{let } (X_i \rightarrow T_i)_{i \geq 1} \text{ in } T
\end{array}
$$

where $l \in \Sigma$ and $X \in \mathit{TVar}$ assuming that TVar is a countably infinite set of type variables.

Abbreviated type expressions can be defined as follows:

$$
\begin{array}{lcl}
T? & = & () \shortmid T \\
T* & = & \text{let } X \rightarrow T \text{ in } T, X \shortmid () \\
T^+ & = & T, T*
\end{array}
$$

Given an environment θ of type variable bindings, the semantics of tree types is given by the denotation function $[\![\cdot]\!]_\theta$:

$$\begin{array}{lcl}
[\![\cdot]\!]. & : & \mathcal{L}_{CFT} \times (TVar \rightarrow 2^{\mathcal{T}^n_\Sigma}) \rightarrow 2^{\mathcal{T}^n_\Sigma} \\
[\![\emptyset]\!]_\theta & = & \emptyset \\
[\![()]\!]_\theta & = & \{()\} \\
[\![X]\!]_\theta & = & \theta(X) \\
[\![l[T])]\!]_\theta & = & \{l'(t) \mid l' \prec l \wedge t \in [\![T]\!]_\theta\} \\
[\![T_1, T_2]\!]_\theta & = & \{t_1, t_2 \mid t_1 \in [\![T_1]\!]_\theta \wedge t_2 \in [\![T_2]\!]_\theta\} \\
[\![T_1 \mid T_2]\!]_\theta & = & [\![T_1]\!]_\theta \cup [\![T_2]\!]_\theta \\
[\![\text{let } (X_i \rightarrow T_i)_{i\geq 1} \text{ in } T]\!]_\theta & = & [\![T]\!]_{lfp(S)}
\end{array}$$

where $\prec$ is a global subtagging relation: a reflexive and transitive relation on labels[5], and $S(\theta') = \theta[X_i \mapsto [\![T_i]\!]_{\theta'}]_{i\geq 1}$. Note that each function S is monotone according to the ordering $\subseteq$ on $TVar \rightarrow 2^{\mathcal{T}^n_\Sigma}$, and thus has a least fixpoint $lfp(S)$.

Types as defined above actually correspond to arbitrary context-free tree types, for which the decision problem for inclusion is known to be undecidable [21]. We impose the additional restriction used in [22] to reduce the expressive power of considered types so that they correspond to regular tree languages. The restriction consists in a simple syntactic condition that allows unguarded (i.e. not enclosed by a label) recursive uses of variables, but restricts them to tail positions[6]. This condition ensures regularity, and we name $\mathcal{L}_{RT}$ the resulting class of regular tree languages. It is well known that deciding inclusion of regular types (i.e. containment of finite tree automata) is in EXPTIME, and the algorithm described in [22] is effective in practice. From an XML point of view, regular tree types form a superset of standards such as XML Schemas and DTDs. We further detail the connection with the widely used DTD standard.

4.1 Document Type Definitions

As they are defined in the W3C recommendation, DTDs [8] are local tree grammars[7], which are strictly less expressive than regular tree types. In the XML terminology, a type expression is often called the *content model*. DTD content models are described by the following syntax:

$$T ::= l \mid T_1 \mid T_2 \mid T_1, T_2 \mid T? \mid T^* \mid T^+ \mid ()$$

where $l \in \Sigma$. From the W3C specification, we see a DTD as a function that associates a content model to each label taken from a subset Σ' of Σ, such that Σ' gathers all labels

[5] Subtagging goes beyond the expressive power of DTDs but a similar notion called "substitution groups" exists in XML Schemas (see [22] for more details on subtagging).

[6] For instance the type "let $(X \rightarrow a[], Y)(Y \rightarrow b[], X \mid ())$ in X" is allowed.

[7] A local tree grammar is a regular tree grammar without *competing* non-terminals. Two non-terminals A and B of a tree grammar are said to compete with each other if one production rule has A in its left-hand side, one production rule has B in its left-hand side, and these two rules share the same terminal symbol in the right-hand side.

$$
\begin{array}{lcl}
\mathcal{B}(\cdot) & : & \mathcal{L}_{RT} \to \mathcal{L}_{BT} \\
\mathcal{B}(\emptyset) & = & \emptyset \\
\mathcal{B}(()) & = & \epsilon \\
\mathcal{B}(X) & = & \mathcal{B}(\theta(X)) \\
\mathcal{B}(l[T]) & = & \text{let } (X_1 \to \mathcal{B}(T))(X_2 \to \epsilon) \text{ in } l(X_1, X_2) \\
\mathcal{B}(T_1 \mid T_2) & = & \mathcal{B}(T_1) \mid \mathcal{B}(T_2) \\
\mathcal{B}(\text{let } (X_i \to T_i)_{i\geq 1} \text{ in } T) & = & \text{let } (X_i \to \mathcal{B}(T_i))_{i\geq 1} \text{ in } \mathcal{B}(T) \\
\mathcal{B}(\emptyset, T) & = & \emptyset \\
\mathcal{B}((), T) & = & \mathcal{B}(T) \\
\mathcal{B}(X, T) & = & \mathcal{B}(\theta(X), T) \\
\mathcal{B}(l[T_1], T_2) & = & \text{let } (X_1 \to \mathcal{B}(T_1))(X_2 \to \mathcal{B}(T_2)) \text{ in } l(X_1, X_2) \\
\mathcal{B}((T_1 \mid T_2), T_3) & = & \mathcal{B}(T_1, T_3) \mid \mathcal{B}(T_2, T_3) \\
\mathcal{B}((T_1, T_2), T_3) & = & \mathcal{B}(T_1, (T_2, T_3)) \\
\mathcal{B}(\text{let } (X_i \to T_i)_{i\geq 1} \text{ in } T, T') & = & \text{let } (X_i \to \mathcal{B}(T_i))_{i\geq 1} \text{ in } \mathcal{B}(T, T')
\end{array}
$$

Figure 12: Binarization of Tree Types.

used in content models. We thus represent the set $\mathcal{L}_{DTD}$ of tree types described by DTDs as follows:

$$
\begin{array}{rcl}
\mathcal{L}_{DTD} \ni T & ::= & l \mid T_1 \mid T_2 \mid T_1, T_2 \mid T? \mid T^* \mid T^+ \mid () \mid \\
& & \text{let } (l_i \to T_i)_{i\geq 1} \text{ in } T
\end{array}
$$

Note that $\mathcal{L}_{DTD} \subseteq \mathcal{L}_{RT}$ is obvious, by associating a unique type variable to each label. In the following, we therefore do not distinguish DTDs from general regular tree types anymore.

4.2 Binarization of Types

In section 2, we used a straightforward isomorphism between binary trees and sequences of unranked trees. There is also an isomorphism between unranked and binary tree types, which follows exactly the same intuition as for trees.

Binary tree types are described by the following syntax:

$$
\begin{array}{rcl}
\mathcal{L}_{BT} \ni T & ::= & \emptyset \mid \epsilon \mid T_1 \mid T_2 \mid l(X_1, X_2) \mid \\
& & \text{let } (X_i \to T_i)_{i\geq 1} \text{ in } T
\end{array}
$$

For any type, there is an equivalent binary type, and vice-versa. We use the translation function shown on Figure 12 (adapted from the one found in [22]) to convert a type into its corresponding binary representation. The function considers the environment $\theta : TVar \to \mathcal{L}_{RT}$ for accessing the type bound to a variable X_i by constructs of the form "let $(X_i \to T_i)_{i\geq 1}$ in T".

4.3 Translation into Mu-Calculus

We now introduce the translation of regular tree types into μ-calculus, which is made simpler by the binary representation of types:

$$\begin{array}{lcl}
[\![\cdot]\!] & : & \mathcal{L}_{BT} \rightarrow \mathcal{L}_{\mu} \\
[\![\emptyset]\!] & = & \bot \\
[\![\epsilon]\!] & = & \bot \\
[\![T_1 \mid T_2]\!] & = & [\![T_1]\!] \vee [\![T_2]\!] \\
[\![l(X_1, X_2)]\!] & = & l \wedge succ_1(X_1) \wedge succ_2(X_2) \\
[\![\text{let } (X_i \rightarrow T_i)_{i \geq 1} \text{ in } T]\!] & = & [\![T]\!][X_i / \mu X_i.[\![T_i]\!]]_{i \geq 1}
\end{array}$$

where $\varphi[X_i/\varphi_i]_{i\geq 1}$ denotes the formula φ in which all occurrences of X_i have been replaced by φ_i[8]. The function $succ.(\cdot)$ takes care of setting the tree frontier:

$$\begin{array}{lcl}
succ.(\cdot) & : & Prog \times TVar \rightarrow \mathcal{L}_{\mu} \\
succ_{\alpha}(X) & = & \begin{cases} [\alpha]\, X & \textit{if } nullable(X) \\ \langle \alpha \rangle\, X & \textit{if not } nullable(X) \end{cases}
\end{array}$$

according to the predicate $nullable(\cdot)$ indicating if a type contains the empty tree:

$$\begin{array}{lcl}
nullable(\cdot) & : & TVar \cup \mathcal{L}_{BT} \rightarrow \{0, 1\} \\
nullable(X) & = & nullable(\theta(X)) \\
nullable(\emptyset) & = & 0 \\
nullable(\epsilon) & = & 1 \\
nullable(l) & = & 0 \\
nullable(T_1 \mid T_2) & = & nullable(T_1) + nullable(T_2) \\
nullable(l(X_1, X_2)) & = & 0 \\
nullable(\text{let } (X_i \rightarrow T_i)_{i \geq 1} \text{ in } T) & = & nullable(T)
\end{array}$$

5 XML Decision Problems

We have translated both XPath over unranked trees, and regular unranked tree types in our unifying $\mathcal{L}_{\mu}$ logic over binary trees. Owing to these embeddings, we now reduce XML decision problems (such as XPath satisfiability, containment, equivalence, overlap, coverage...) to satisfiability in $\mathcal{L}_{\mu}$.

We first introduce some simplified notations. For an XPath expression $e \in \mathcal{L}_{\text{XPath}}$, we note φ_e the translated formula $E^{\rightarrow}[\![e]\!](\top) \in \mathcal{L}_{\mu}$. Furthermore, we note $\mathcal{T}$ the set of trees: by default, $\mathcal{T} = \mathcal{T}_{\Sigma}^{n}$, and whenever an optional DTD $d \in \mathcal{L}_{DTD}$ is specified $\mathcal{T} = [\![d]\!]_{\emptyset}$. Finally, we note $\varphi_{\mathcal{T}}$ the $\mathcal{L}_{\mu}$ embedding of the tree language $\mathcal{T}$. In the absence of DTDs $\varphi_{\mathcal{T}} = \top$, and $\varphi_{\mathcal{T}} = [\![\mathcal{B}(d)]\!]$ in the presence of a DTD $d \in \mathcal{L}_{DTD}$.

[8] In practice, this formula is not built in memory but instead represented using a valuation of type variables in order to avoid exponential blow-ups caused by the theoretical duplication of subformulae.

Decision Problem	Satisfiability	Containment
Input	$e \in \mathcal{L}_{\text{XPath}}$	$e_1, e_2 \in \mathcal{L}_{\text{XPath}}$
Optional Input	$d \in \mathcal{L}_{DTD}$	$d \in \mathcal{L}_{DTD}$
Definition	$\exists t \in \mathcal{T} : \exists x \in t :$ $\mathcal{S}_e[\![e]\!]^t_x \neq \emptyset$	$\forall t \in \mathcal{T}, \forall x \in t,$ $\mathcal{S}_e[\![e_1]\!]^t_x \subseteq \mathcal{S}_e[\![e_2]\!]^t_x$
$\varphi_{\text{tested}} \in \mathcal{L}_\mu$	φ_e	$\varphi_{e_1} \wedge \neg\varphi_{e_2}$

Decision Problem	Overlap	Coverage
Input	$e_1, e_2 \in \mathcal{L}_{\text{XPath}}$	$e_1, e_2, ..., e_n \in \mathcal{L}_{\text{XPath}}$
Optional Input	$d \in \mathcal{L}_{DTD}$	$d \in \mathcal{L}_{DTD}$
Definition	$\forall t \in \mathcal{T}, \forall x \in t,$ $\mathcal{S}_e[\![e_1]\!]^t_x \cap \mathcal{S}_e[\![e_2]\!]^t_x \neq \emptyset$	$\forall t \in \mathcal{T}, \forall x \in t,$ $\mathcal{S}_e[\![e_1]\!]^t_x \subseteq \bigcup_{2<i<n} \mathcal{S}_e[\![e_i]\!]^t_x$
$\varphi_{e_1} \wedge \neg\varphi_{e_2}$	$\varphi_{e_1} \wedge \varphi_{e_2}$	$\varphi_{e_1} \wedge \bigwedge_{2<i<n} \neg\varphi_{e_i}$

Table 1: Some XPath Decision Problems and their μ-Calculus Counterpart.

Table 1 presents how several decision problems needed in applications can be expressed in terms of a $\mathcal{L}_\mu$ formulae φ_{tested}.

Since we need to enforce the finite binary tree model property (as seen in Section 2.2), we formulate decision problems from the root, and the actually tested formula becomes:

$$\varphi_{\text{root}} \wedge \varphi_{\text{ft}} \wedge (\varphi_{\mathcal{T}} \wedge \mu X.\varphi_{\text{tested}} \vee \langle 1\rangle\, X \vee \langle 2\rangle\, X)^{\text{FBT}} \tag{1}$$

Intuitively, the fixpoint is introduced for "plunging" XPath navigation performed by φ_{tested} at any location in the tree. It is for example necessary for relative XPath expressions that involve backward navigation in the tree.

Note that for the containment problem, we actually test the unsatisfiability of φ_{tested}. Indeed, checking that an XPath expression e_1 is contained into another expression e_2 consists in checking that the implication $\varphi_{e_1} \Rightarrow \varphi_{e_2}$ holds for all trees. In other terms, there exists no tree for which the results of e_1 are not included in those of e_2, i.e. the negated implication $\varphi_{e_1} \wedge \neg\varphi_{e_2} = \varphi_{\text{tested}}$ is unsatisfiable.

The XPath equivalence problem can be tested by two successive and separate containment checks.

It is important to note that formula (1) is always alternation-free since both embeddings of XPath and tree types produce alternation-free formulae, and the negation of an alternation free sentence remains alternation-free. In practice, negated sentences introduced by XPath embeddings are turned into negation normal form, by applying the rules given on Figure 4.

6 Preliminary Experiments

The objective of the section aims at testing the practical performance of our method. The proposed approach has been fully implemented. A compiler takes XPath expressions as

input, and translates them into $\mathcal{L}_\mu$ formulae. Another compiler takes regular tree types as input (DTDs) and outputs their $\mathcal{L}_\mu$ translation. The formula of a particular decision problem is then composed, normalized and solved.

The $\mathcal{L}_\mu$ satisfiability solver is based on the tableau method described in [35]. It is specialized for the alternation-free μ-calculus with backward modalities. The time complexity of the decision procedure is $2^{O(n \log n)}$ with respect to the length n of the given formula, which is more efficient than the complexity for the whole μ-calculus with backward modalities [38]. Whenever the containment does not hold, the solver outputs a counter-example XML tree.

We carried out several testing scenarios[9], but for a lack of space we present only a few of them. First, we used an XPath benchmark [16] whose goal is to cover XPath features by gathering a significant variety of XPath expressions met in real-world applications. In this first test series, we do not consider types yet, and only focus on the XPath containment problem, since its logical formulation (presented in Section 5) is the most complex, as it requires the logic to be closed under negation. The first test series consists in finding the relation holding for each pair of queries from the benchmark. This means checking the containment of each query of the benchmark against all the others. We note $q_i \subseteq q_j$ whenever the query q_i is included into the query q_j. Comparisons of two queries q_i and q_j may yield to three different results:

1. $q_i \subseteq q_j$ and $q_j \subseteq q_i$, the queries are semantically equivalent, we note $q_i \equiv q_j$
2. $q_i \subseteq q_j$ but $q_j \not\subseteq q_i$, we denote this by $q_i \subset q_j$ or alternatively by $q_j \supset q_i$
3. $q_i \not\subseteq q_j$ and $q_j \not\subseteq q_i$, queries are not related, we note $q_i \not\sim q_j$

Queries are presented on Figure 13. Corresponding results together with running times of the decision procedure are summarized on Table 2. Times reported in milliseconds correspond to the actual running time of the μ-calculus satisfiability solver without the extra time spent for parsing XPath nor the (linear) cost of the translation into μ-calculus. Obtained results show that all tests are solved in several milliseconds. This suggests that XPath expressions used in real-world scenarios can be efficiently handled in practice.

As a second test series, we compare expressions found in research papers on the containment of XPath expressions. Figure 14 presents the expressions we collected. For this set of expressions, the tree pattern homomorphism technique [30] returns false negatives, whereas our approach is complete. Table 3 shows the results obtained with our system. This suggests that our system is able to reasonably handle containment instances which are difficult to solve using other techniques.

The third test series aims at showing the effectiveness of the system for XPath decision problems in presence of DTDs. We used a small recursive DTD (given on Figure 15), and real-world DTDs of the SMIL [19] and XHTML [18] standards. Table 4 gives hints on the respective complexity of each DTD by presenting the number of symbols used (alphabet

[9]Experiments have been conducted on a Pentium 4, 3 Ghz, with 512Mb of RAM, running Eclipse on Windows XP.

q_1 /site/regions/*/item
q_2 /site/closedauctions/closedauction/annotation/description/parlist/listitem/text/keyword
q_3 //keyword
q_4 /descendant-or-self::listitem/descendant-or-self::keyword
q_5 /site/regions/*/item[parent::namerica or parent::samerica]
q_6 //keyword/ancestor::listitem
q_7 //keyword/ancestor-or-self::mail
q_8 /site/regions/namerica/item | /site/regions/samerica/item
q_9 /site/people/person[address and (phone or homepage)]

Figure 13: XPath Queries Taken from the XPathmark Benchmark.

Relation	Time (ms)	
	$\subseteq$	$\supseteq$
$q_1 \not\propto q_2$	8	2
$q_1 \not\propto q_3$	2	2
$q_1 \not\propto q_4$	4	2
$q_1 \supset q_5$	6	7
$q_1 \not\propto q_6$	4	3
$q_1 \not\propto q_7$	5	2
$q_1 \supset q_8$	4	11
$q_1 \not\propto q_9$	6	6
$q_2 \subset q_3$	12	3
$q_2 \subset q_4$	8	6
$q_2 \not\propto q_5$	14	13
$q_2 \not\propto q_6$	12	5
$q_2 \not\propto q_7$	9	6
$q_2 \not\propto q_8$	16	14
$q_2 \not\propto q_9$	14	11
$q_3 \supset q_4$	1	2
$q_3 \not\propto q_5$	3	4
$q_3 \not\propto q_6$	1	3

Relation	Time (ms)	
	$\subseteq$	$\supseteq$
$q_3 \not\propto q_7$	2	2
$q_3 \not\propto q_8$	4	7
$q_3 \not\propto q_9$	3	5
$q_4 \not\propto q_5$	4	5
$q_4 \not\propto q_6$	1	2
$q_4 \not\propto q_7$	2	2
$q_4 \not\propto q_8$	4	7
$q_4 \not\propto q_9$	3	4
$q_5 \not\propto q_6$	4	4
$q_5 \not\propto q_7$	5	4
$q_5 \equiv q_8$	12	12
$q_5 \not\propto q_9$	8	8
$q_6 \not\propto q_7$	2	2
$q_6 \not\propto q_8$	5	7
$q_6 \not\propto q_9$	4	5
$q_7 \not\propto q_8$	6	8
$q_7 \not\propto q_9$	4	6
$q_8 \not\propto q_9$	11	10

Table 2: Results and Total Computation Times.

e_1 $/a[.//b[c/*//d]/b[c//d]/b[c/d]]$
e_2 $/a[.//b[c/*//d]/b[c/d]]$

e_3 $a[b]/*/d/*/g$
e_4 $a[b]/(b|c)/d/(e|f)/g$
e_5 $a[b]/b/d/e/g|a/b/d/f/g$

e_6 $a/b/s//c/b/s/c//d$
e_7 $a//b/*/c//*/d$

e_8 $a[b/e][b/f][c]$
e_9 $a[b/e][b/f]$

e_{10} /descendant::editor[parent::journal]
e_{11} /descendant-or-self::journal/child::editor

Figure 14: Instances Found in Research Papers.

Relation	Time (ms)	
	$\subseteq$	$\supseteq$
$e_1 \subset e_2$	11	10
$e_3 \supset e_4$	10	14
$e_3 \supset e_5$	8	17
$e_4 \supset e_5$	13	19
$e_6 \subset e_7$	24	23
$e_8 \subset e_9$	1	2
$e_{10} \equiv e_{11}$	1	1

Table 3: Results and Running Times.

```
<!ELEMENT people (person*)>
<!ELEMENT person (name,birthdate?,gender?,children?)>
<!ELEMENT name (firstname+, lastname) >
<!ELEMENT firstname (#PCDATA) >
<!ELEMENT lastname (#PCDATA) >
<!ELEMENT birthdate (#PCDATA) >
<!ELEMENT gender (#PCDATA) >
<!ELEMENT children (person+) >
```

Figure 15: (People.dtd) A Simple Recursive DTD.

Small and Real-World DTDs	Symbols	Type Variables	Binary Type Variables
People.dtd (given in appendix)	8	15	11
SMIL 1.0 [19]	19	13	29
XHTML 1.0 Strict [18]	77	104	254

Table 4: DTDs Used in Practical Experiments.

size) and the number of grammar production rules (type variables) in the unranked and binary representations.

For each DTD, we built several XPath decision problems using the expressions shown on Figure 16. Some decision problems and their results are presented on Table 5. The system performs well for the relatively small "People" and "SMIL" DTDs, even if they are recursive. The satisfiability test for p_5 illustrates an additional benefit of the method which outputs a valid SMIL document as a satisfying example:

```
<smil>
  <head>
    <switch>
      <layout/>
    </switch>
    <meta/>
  </head>
  <body/>
</smil>
```

We use the XHTML DTD to test the scalability of the approach. The system can prove properties such as links cannot be nested, image tags must be leaves, and that every element is either in the header or in the body of an XHTML document. We observe that the time needed is significantly more important, but proving these properties remains valuable and practically feasible, especially for static analysis purposes where such operations are performed at compile-time. Interestingly, a large amount of time is spent in the μ-loop detection performed by the solver for avoiding potential cycles and infinite paths in the case of finite recursion [35]. Our approach may thus benefit from being further developed by taking advantage of XML peculiarities at the solver level.

These preliminary measurements shed light, for the first time, on the cost of solving XPath decision problems in practice. They strengthen the hope for an effective static analysis of standard XML transformations in the near future.

7 Related Work

From a theoretical perspective, several logical formalisms have been used to chart the decidability frontier of XPath decision problems. Some EXPTIME upper bounds are already known for satisfiability and containment of specific subsets of our XPath fragment.

p_1 people/*
p_2 //person
p_3 /descendant-or-self/people/person
p_4 //children/person

p_5 switch/layout
p_6 smil/head//layout
p_7 smil/head//layout[ancestor::switch]

p_8 descendant::a[ancestor::a]
p_9 /descendant::*
p_{10} html/(head | body)
p_{11} html/head/descendant::*
p_{12} html/body/descendant::*
p_{13} //img
p_{14} //img[not *]

Figure 16: XPath Expressions used with DTDs.

XPath Decision Problem	Instance	DTD	Answer	Time (ms)
Containment	$p_1 \subseteq p_2$	People.dtd	true	32
Coverage	$p_2 \subseteq p_3 \cup p_4$	People.dtd	true	41
Satisfiability	p_5	SMIL 1.0	true	80
Overlap	$p_5 \cap p_6 \neq \emptyset$	SMIL 1.0	false	74
Containment	$p_6 \subseteq p_7$	SMIL 1.0	false	90
Satisfiability	p_8	XHTML 1.0	false	98520
Coverage	$p_9 \subseteq p_{10} \cup p_{11} \cup p_{12}$	XHTML 1.0	true	181872
Containment	$p_{13} \subseteq p_{14}$	XHTML 1.0	true	154931

Table 5: Some Decision Problems and Corresponding Results.

Close in spirit to our paper is the constructive connection between XPath and formal logics, which is actively studied [29, 6, 5]. In particular, [29] characterizes XPath in terms of extensions of Computational Tree Logic (CTL) [10], which is equivalent to first order logic (FO) over tree structures [28, 5] and whose satisfiability is in EXPTIME. Authors of [30] first observed that a fragment of XPath can be embedded in CTL. However, regular tree languages are not fully captured by such FO variants [7]. The work found in [2] proposes a variant of Propositional Dynamic Logic (PDL) [15] with a similar EXPTIME complexity for reasoning about ordered trees, but whose exact expressive power is still under study.

The complexity of XPath satisfiability in the presence of DTDs is studied in [6]. XPath containment has specifically attracted a lot of research attention [3, 11, 30, 32, 33, 40, 41]. Prior work concentrated on various combinations of the previous factors for obtaining complexity results (see [33] for an overview). Specifically, the focus was given to restricted positive XPath subfragments without upward axes. In particular, [32] proves an EXPTIME upper-bound for containment (in the presence of DTDs) of queries containing the "child" and "descendant" axes, and union of paths. [11] considers XPath containment in the presence of DTDs and simple XPath integrity constraints (SXICS). They obtain that this problem is undecidable in general and in the presence of bounded SXICs and DTDs. Containment for the fragment $\mathrm{XP}^{\{*,//,[\,]\}}$ is shown to be coNP-complete in [30], where the containment mapping technique relies on a polynomial time tree homomorphism algorithm, which gives a sufficient but not necessary condition for containment of $\mathrm{XP}^{\{*,//,[\,]\}}$ in general. Additionally, the containment problem is shown to be in EXPTIME for the fragments $\mathrm{XP}^{\{//,[\,]\}}$, $\mathrm{XP}^{\{//,[\,],|\}}$, $\mathrm{XP}^{\{//,|\}}$ in the presence of DTDs in [41].

Compared to all these previous works, the XPath fragment we consider is far more complete and much more realistic. We also present a single unifying logical framework in which all major XPath features but also regular tree types fit together. Moreover, our framework yields effective decision procedures usable in practice for real world scenarios (whereas no implementation has been reported in prior work). Finally, from a theoretical perspective, we also see the connection between XML and the μ-calculus as a much simpler way of deriving the EXPTIME upper-bounds of decision problems needed in applications.

8 Conclusion

In this paper, we proposed a new logical approach for XPath decision problems. XPath queries and regular tree types are translated into the μ-calculus. XML decision problems are expressed in terms of formulae in this logic, then decided using a state-of-the-art decision procedure for μ-calculus satisfiability. This paper makes several contributions.

First, we propose a specific variant of the μ-calculus, namely the alternation-free modal μ-calculus with backward modalities, as the appropriate logic for modeling XML data, XPath queries and XML types. As a valuable outcome, we show a linear translation of XPath in the μ-calculus, and a compatible embedding of regular tree types.

Second, we take advantage of these translations to reduce several XML decision problems to satisfiability in $\mathcal{L}_\mu$. We obtain effective EXPTIME decision procedures, usable in practice.

The considered XPath fragment includes union, intersection, path composition together with all forward and backward axes, branching, boolean connectives, wildcards, and negation, in the presence or absence of DTDs. This fragment is far more complete than other fragments addressed in previous studies.

The global proposed approach has been implemented. We provide practical experiments and detailed results that corroborate our claim that this approach is efficient in practice for real-world XPath expressions and DTDs.

Eventually, an additional advantage of this technique is to allow generation of XML tree examples when the containment does not hold. We believe this makes our method of special interest for many applications including debuggers, or applications that can benefit from a precise reporting during static analysis stages.

One direction of future work consists in specifically tuning the μ-calculus satisfiability solver for XML, in order to obtain even more efficient decision procedures. Incorporating XML peculiarities directly in the core of the μ-calculus solver may yield even more efficient decision procedures for XML. A further characterization of the $\mathcal{L}_\mu$ fragment actually needed for XPath and XML types might be used for obtaining lower complexity bounds for XML decision problems. It could also be used for characterizing XPath evaluation, which is reduced to $\mathcal{L}_\mu$ model-checking by our translations. Finally, another direction is to push the "decidability envelope" even further by considering XPath data values, for which we do not know any semantics-preserving translation into an appropriate formalism yet.

References

[1] S. Abiteboul and V. Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3):428–452, 1999.

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

[3] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of tree pattern queries. *SIGMOD Record*, 30(2):497–508, 2001.

[4] A. Arnold and D. Niwinski. Fixed point characterization of weak monadic logic definable sets of trees. In *Tree Automata and Languages*, pages 159–188. North-Holland, Amsterdam, Netherlands, 1992.

[5] P. Barceló and L. Libkin. Temporal logics over unranked trees. In *LICS '05: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, pages 31–40, New York, NY, USA, 2005. IEEE Computer Society.

[6] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *PODS '05: Proceedings of the twenty-fourth ACM Symposium on Principles of Database Systems*, pages 25–36, New York, NY, USA, 2005. ACM Press.

[7] M. Benedikt and L. Segoufin. Regular tree languages definable in FO. In *STACS '05: Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *LNCS*, pages 327–339, London, UK, 2005. Springer-Verlag.

[8] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (third edition), W3C recommendation, February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/.

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

[10] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, volume 131 of *LNCS*, pages 52–71, London, UK, 1981. Springer-Verlag.

[11] A. Deutsch and V. Tannen. Containment of regular path expressions under integrity constraints. In *KRDB '01: Proceedings of the 8th International Workshop on Knowledge Representation meets Databases*, volume 45 of *CEUR Workshop Proceedings*, pages 1–11, ceur-ws.org, 2001. CEUR.

[12] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[13] D. C. Fallside and P. Walmsley. XML Schema part 0: Primer second edition, W3C recommendation, October 2004. http://www.w3.org/TR/xmlschema-0/.

[14] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML querying with security views. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 587–598, New York, NY, USA, 2004. ACM Press.

[15] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[16] M. Franceschet. XPathMark - an XPath benchmark for XMark generated data. In *XSYM '05: Proceedings of The Third International Symposium on Database and XML Technologies*, volume 3671 of *LNCS*, pages 129–143, London, UK, 2005. Springer-Verlag.

[17] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005.

[18] T. W. H. W. Group. XHTML 1.0 the extensible hypertext markup language, W3C recommendation, January 2000. http://www.w3.org/TR/xhtml1/.

[19] T. W. S. M. W. Group. Synchronized multimedia integration language (SMIL) 1.0 specification, W3C recommendation, June 1998. http://www.w3.org/TR/REC-smil/.

[20] A. Grzegorczyk. Some classes of recursive functions. *Rozprawy Matematyczne*, 4:1–45, 1953.

[21] J. E. Hopcroft, R. Motwani, Rotwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[22] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for xml. *ACM Transactions on Programming Languages and Systems*, 27(1):46–90, 2005.

[23] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[24] D. Kozen. A finite model theorem for the propositional mu-calculus. *Studia Logica*, 47(3):233–241, 1988.

[25] O. Kupferman and M. Vardi. The weakness of self-complementation. In *Proc. 16th Symp. on Theoretical Aspects of Computer Science*, volume 1563 of *LNCS*, pages 455–466, London, UK, 1999. Springer-Verlag.

[26] M. Y. Levin and B. C. Pierce. Type-based optimization for regular patterns. In *DBPL '05: Proceedings of the 10th International Symposium on Database Programming Languages*, volume 3774 of *Lecture Notes in Computer Science*, London, UK, August 2005. Springer-Verlag.

[27] W. Martens and F. Neven. Frontiers of tractability for typechecking simple XML transformations. In *PODS '04: Proceedings of the twenty-third ACM Symposium on Principles of Database Systems*, pages 23–34, New York, NY, USA, 2004. ACM Press.

[28] M. Marx. Conditional XPath, the first order complete XPath dialect. In *PODS '04: Proceedings of the twenty-third ACM Symposium on Principles of Database Systems*, pages 13–22, New York, NY, USA, 2004. ACM Press.

[29] M. Marx. XPath with conditional axis relations. In *Proceedings of the 9th International Conference on Extending Database Technology*, volume 2992 of *LNCS*, pages 477–494, London, UK, January 2004. Springer-Verlag.

[30] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004.

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

[32] F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, volume 2572 of *LNCS*, pages 315–329, London, UK, 2003. Springer-Verlag.

[33] T. Schwentick. XPath query containment. *SIGMOD Record*, 33(1):101–109, 2004.

[34] G. Sur, J. Hammer, and J. Siméon. Updatex - an XQuery-based language for processing updates in XML. In *PLAN-X 2004: Proceedings of the International Workshop on Programming Language Technologies for XML, Venice, Italy*, volume NS-03-4 of *BRICS Notes Series*, pages 40–53, Aarhus, Denmark, January 2004. BRICS.

[35] Y. Tanabe, K. Takahashi, M. Yamamoto, A. Tozawa, and M. Hagiya. A decision procedure for the alternation-free two-way modal μ-calculus. In *TABLEAUX '05: Proceedings of the 14th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 3702 of *LNCS*, pages 277–291, London, UK, September 2005. Springer-Verlag.

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

[37] A. Tozawa. Towards static type checking for XSLT. In *DocEng '01: Proceedings of the 2001 ACM Symposium on Document Engineering*, pages 18–27, New York, NY, USA, 2001. ACM Press.

[38] M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 628–641, London, UK, 1998. Springer-Verlag.

[39] P. Wadler. Two semantics for XPath. Internal Technical Note of the W3C XSL Working Group, http://homepages.inf.ed.ac.uk/wadler/papers/xpath-semantics/xpath-semantics.pdf, January 2000.

[40] P. T. Wood. On the equivalence of XML patterns. In *CL '00: Proceedings of the First International Conference on Computational Logic*, volume 1861 of *LNCS*, pages 1152–1166, London, UK, 2000. Springer-Verlag.

[41] P. T. Wood. Containment for XPath fragments under DTD constraints. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, volume 2572 of *LNCS*, pages 300–314, London, UK, August 2003. Springer-Verlag.

Contents

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

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

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