Comparing XML Path Expressions

Pierre Genevès, Nabil Layaïda

Comparing XML Path Expressions

Pierre Genevès
INRIA Rhône-Alpes
France
pierre.geneves@inria.fr

Nabil Layaïda
INRIA Rhône-Alpes
France
nabil.layaida@inria.fr

ABSTRACT

XPath is the standard declarative language 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 comparisons such as equivalence (whether two queries always return the same result), and containment (whether for any tree the result of a particular query is included in the result of a second one).

XPath decision problems have attracted a lot of research attention, especially for studying the computational complexity of various XPath fragments. However, what is missing at present is the constructive use of an expressive logic which would allow capturing these decision problems, while providing practically effective decision procedures.

In this paper, we propose a logic-based framework for the static analysis of XPath. Specifically, we propose the alternation free modal μ-calculus with converse as the appropriate logic for effectively solving XPath decision problems. We present a translation of a large XPath fragment into μ-calculus, together with practical experiments on the containment using a state-of-the-art EXPTIME decision procedure for μ-calculus satisfiability. These preliminary experiments shed light, for the first time, on the cost of checking the containment in practice. We believe they reveal encouraging results for further static analysis of XML transformations.

Categories and Subject Descriptors

D.3.1 [**Formal Definitions and Theory**]: Semantics; F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Mechanical verification

General Terms

Algorithms, Languages, Standardization, Theory

Keywords

XPath, Analysis, Experimentation

DocEng'06, October 10–13, 2006, Amsterdam, The Netherlands.

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 [33] and XML access control [14].

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, 24].

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 [25, 36].

A variety of factors contribute to its complexity 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.

XPath containment has recently attracted a lot of research attention [3, 11, 29, 31, 32, 39, 40]. Prior work concentrated on various combinations of the previous factors for obtaining complexity results (see [32] for an overview). Specifically, the focus was given to restricted XPath fragments, namely positive XPath subfragments without upward axes. In particular, [31] proves an EXPTIME upper-bound for containment (in the presence of DTDs) of queries containing the "child" and "descendant" axes, and union of paths.

From the results of [6, 32], we know that the combination of some previous factors with data values may lead to undecidability of 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 with respect to containment.

Close in spirit to our paper is the constructive connection between XPath and formal logics, which is actively studied [27, 6, 5]. In particular, [27] characterizes XPath in terms of extensions of Computational Tree Logic (CTL) [10], which is equivalent to first order logic (FO) over tree structures [26, 5] and whose satisfiability is in EXPTIME. Authors of [29] first observed that a fragment of XPath can be embedded in CTL. This was further developed in [18]. 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. Regular tree languages are not fully captured by FO variants [8].

The most expressive logic considered for XML is Monadic Second Order Logic (MSO), 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) [35, 12]. From [4, 23], we know that WS2S is exactly as expressive as the alternation-free fragment (AFMC) of the propositional modal μ-calculus introduced in [21]. 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 and PDL. Furthermore, the work in [37] adds converse programs to the propositional modal μ-calculus and shows that the resulting logic still admits an EXPTIME decision procedure for satisfiability.

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

[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.

A translation of XPath in μ-calculus is therefore of central interest. Such a translation can also be seen as a much simpler way of deriving the EXPTIME upper-bound for the XPath containment problem. It also allows to derive a linear upper-bound for XPath evaluation (model-checking) owing to [28], thus matching the best complexity of monadic datalog [17], as pointed in [5].

In this paper, we propose the alternation free modal μ-calculus with converse as the appropriate logic for effectively solving XPath decision problems. We present a linear translation of a large XPath fragment into μ-calculus. This yields practical procedures usable for answering XPath decision problems needed in applications such as:

- overlap ("is the intersection of two expressions non-empty?"), that corresponds to a simple logical cunjunction in mu-calculus;
- coverage ("is an expression contained in the union of several expressions?"), that can be reduced to containment by merging the right hand side expressions into a single disjunctive expression;
- satisfiability[3];
- equivalence (defined as the mutual containment of two expressions, checked separately);
- containment, whose logical formulation (presented in Section 3.3) is the most complex, as it requires the logic to be closed under negation.

We therefore build on our translation and show how the containment can be effectively decided in practice. Using a state-of-the-art EXPTIME decision procedure for μ-calculus satisfiability, we give experimental results on the containment of realistic XPath queries. These results strengthen the hope for an effective static analysis of XML transformations in the near future.

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; we present experimental results on the fundamental XPath containment problem in Section 4, before discussing related work in Section 5 and concluding in Section 6.

2. A LOGIC FOR XML

We consider XML documents as finite ordered trees of unbounded depth and arity, labeled with symbols taken from an alphabet Σ. It is well-known that such unranked trees can be encoded as binary trees, without loss of generality [30]. The bijective mapping we use is illustrated on Figure 2. We now introduce the logic we propose for reasoning over these structures.

[3]Note that XPath Satisfiability is known to be in PSPACE [6], whereas our approach is designed to address a larger range of problems such as the more complex containment. Thus, although our system can be used for deciding XPath satisfiability and yield acceptable results in practice, it may not be optimal w.r.t complexity for this specific decision problem.

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

Figure 1: XPath Abstract Syntax.

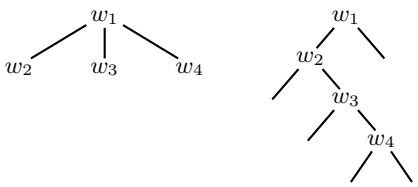

Figure 2: N-ary and Binary Tree Representations.

2.1 The μ-Calculus

The *propositional μ-calculus* is a propositional modal logic extended with least and greatest fixpoint operators [21]. A *signature* Ξ for the μ-calculus consists of a set *Prop* of atomic propositions, a set *Var* of propositional variables, and a set *FProg* 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 converse[4] [37] 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. We note *Prog* the set $\mathit{FProg} \cup \{\overline{a} \mid a \in \mathit{FProg}\}$. This is the only difference between the propositional μ-calculus that lacks converse programs. It is important to note that the addition of converse programs preserves the EXPTIME upper bound for the satisfiability problem [37].

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

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

where $p \in \mathit{Prop}$, $X \in \mathit{Var}$ and α is a program. Note that X should not occur negatively in $\mu X.\varphi$ and in $\nu X.\varphi$

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

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

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

Figure 3: Semantics of the μ-Calculus.

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

Figure 4: Dualities for Positive Normal Form.

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 positive normal form. We say that a formula is in *positive normal form* if and only if all negations in the formula appear only before atomic propositions. Every formula is equivalent to a formula in positive normal form [21], 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 positive normal form.

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

We recall that a $\mathcal{L}_\mu$ formula φ in positive normal form is *alternation-free* whenever the following condition holds[5]: 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.

[5]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.

2.2 XML Trees and 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 [22]. Unfortunately, the introduction of converse programs causes the loss of the finite model property [37]. 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 navigation in 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 [34]. A Kripke structure $T = \langle W, R, L\rangle$ is a finite binary tree if it satisfies the following conditions:

(1) W is finite

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

(3) $\prec_{\text{fc}}$ and $\prec_{\text{ns}}$ 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

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

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:

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

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. $\varphi_{\text{root}} \wedge \varphi_{\text{ft}}$ 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 $\langle a\rangle \psi$ are replaced by $\langle a\rangle \top \wedge [a] \psi^{\text{FBT}}$. This replacement is enough to enforce conditions (2) and (3).

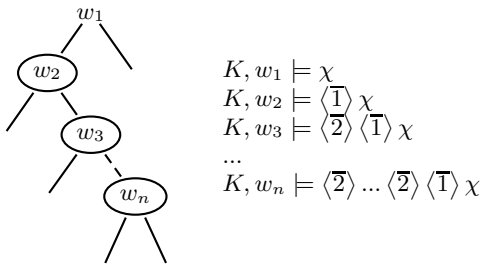

Figure 7: Intuition for the "child" Axis.

PROPOSITION 2.1. *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 [34]. 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 2.1 gives the adequate framework for formulating decision problems on XML structures in terms of a μ-calculus formula.

3. XPATH QUERIES

In this section, we explain how an XPath expression can be translated into an equivalent formula in $\mathcal{L}_\mu$. The translation adheres to XPath formal semantics (given in appendix): 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 Logical Translation 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.

3.2 Logical Translation of Expressions

The translation of XPath expressions into μ-calculus is given on Figure 8. It is formally expressed as a translation

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

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

$$
\begin{array}{lcl}
A^{\rightarrow}[\![\cdot]\!](\cdot) & : & \mathit{Axis} \to \mathcal{L}_\mu \to \mathcal{L}_\mu \\
A^{\rightarrow}[\![\text{self}]\!](\chi) & = & \chi \\
A^{\rightarrow}[\![\text{following-sibling}]\!](\chi) & = & \mu Z.\, \langle\overline{2}\rangle\, \chi \vee \langle\overline{2}\rangle\, Z \\
A^{\rightarrow}[\![\text{child}]\!](\chi) & = & \mu Z.\, \langle\overline{1}\rangle\, \chi \vee \langle\overline{2}\rangle\, Z \\
A^{\rightarrow}[\![\text{descendant}]\!](\chi) & = & \mu Z.\, \langle\overline{1}\rangle\, (\chi \vee Z) \vee \langle\overline{2}\rangle\, Z \\
A^{\rightarrow}[\![\text{descendant-or-self}]\!](\chi) & = & \mu Z.\chi \vee \mu Y.\, \langle\overline{1}\rangle\, (Y \vee Z) \vee \langle\overline{2}\rangle\, Y \\
A^{\rightarrow}[\![\text{parent}]\!](\chi) & = & \langle 1\rangle\, \mu Z.\chi \vee \langle 2\rangle\, Z \\
A^{\rightarrow}[\![\text{ancestor}]\!](\chi) & = & \langle 1\rangle\, \mu Z.\chi \vee \langle 1\rangle\, Z \vee \langle 2\rangle\, Z \\
A^{\rightarrow}[\![\text{ancestor-or-self}]\!](\chi) & = & \mu Z.\chi \vee \langle 1\rangle\, \mu Y.Z \vee \langle 2\rangle\, Y \\
A^{\rightarrow}[\![\text{preceding-sibling}]\!](\chi) & = & \mu Z.\, \langle 2\rangle\, \chi \vee \langle 2\rangle\, Z \\
A^{\rightarrow}[\![\text{following}]\!](\chi) & = & A^{\rightarrow}[\![\text{descendant-or-self}]\!](A^{\rightarrow}[\![\text{following-sibling}]\!](A^{\rightarrow}[\![\text{ancestor-or-self}]\!](\chi))) \\
A^{\rightarrow}[\![\text{preceding}]\!](\chi) & = & A^{\rightarrow}[\![\text{descendant-or-self}]\!](A^{\rightarrow}[\![\text{preceding-sibling}]\!](A^{\rightarrow}[\![\text{ancestor-or-self}]\!](\chi)))
\end{array}
$$

Figure 6: Translation of XPath Axes.

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

Figure 8: Translation of Expressions.

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

Figure 9: Translation of Paths.

$$
\begin{array}{lcl}
Q^{\leftarrow}[\![\cdot]\!](\cdot) & : & \mathit{Qualifier} \to \mathcal{L}_\mu \to \mathcal{L}_\mu \\
Q^{\leftarrow}[\![q_1 \text{ and } q_2]\!](\chi) & = & Q^{\leftarrow}[\![q_1]\!](\chi) \wedge Q^{\leftarrow}[\![q_2]\!](\chi) \\
Q^{\leftarrow}[\![q_1 \text{ or } q_2]\!](\chi) & = & Q^{\leftarrow}[\![q_1]\!](\chi) \vee Q^{\leftarrow}[\![q_2]\!](\chi) \\
Q^{\leftarrow}[\![\text{not } q]\!](\chi) & = & \neg\, Q^{\leftarrow}[\![q]\!](\chi) \\
Q^{\leftarrow}[\![p]\!](\chi) & = & P^{\leftarrow}[\![p]\!](\chi)
\end{array}
$$

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

Figure 10: Translation of Qualifiers.

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. We use a fresh atomic proposition named φ_{context} for distinguishing context nodes.

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 appendix). 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 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 left. This differs from the denotational semantics given in appendix in which the formal semantics functions for paths and qualifiers are mutually recursive (and cause naive implementations to be unnecessarily complex, as pointed out by [19]). Translations of paths inside qualifiers are also given on Figure 10. They

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

Figure 11: Symmetry of Axes inside Qualifiers.

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 φ_{context} 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 φ_{context} modeling context 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, we relate our translations in $\mathcal{L}_\mu$ to the XPath denotational semantics given by Wadler [38] (recalled in appendix), denoted by $\mathcal{S}_e[\![\cdot]\!]$.

PROPOSITION 3.1. *Let T' be an XML tree and e an XPath expression, then for all $y' \in T'$, the followings are equivalent:*

- *There exists $x' \in T'$ such that $y' \in \mathcal{S}_e[\![e]\!]^{T'}_{x'}$*
- *$T, y \models \varphi_{root} \wedge \varphi_{ft} \wedge (E^{\rightarrow}[\![e]\!](\varphi_{context}))^{FBT}$*

where y is the counterpart of y' in the binary tree representation T of T'.

The previous result links XPath decision problems to satisfiability in $\mathcal{L}_\mu$. The proof is done by a straightforward structural induction that "peels off" the compositional layers of each set of rules. Its detailed development is beyond the scope of this paper. Instead we devote the remaining part of the paper to how feasible is to use these translations in practice, and specifically for the containment problem.

3.3 The Containment Formula

Checking whether an XPath expression e_1 is contained into an expression e_2 consists in checking the logical implication of their translations $\varphi_{e_1} = E^{\rightarrow}[\![e_1]\!](\varphi_{\text{context}})$ and $\varphi_{e_2} = E^{\rightarrow}[\![e_2]\!](\varphi_{\text{context}})$ in $\mathcal{L}_\mu$:

$$\varphi_{e_1} \Rightarrow \varphi_{e_2} \tag{1}$$

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 negation of (1), $\varphi_{e_1} \wedge \neg\varphi_{e_2}$, is unsatisfiable.

To enforce the finite binary tree model property (as seen in Section 2.1), we formulate the problem from the root and obtain:

$$\varphi_{\text{root}} \wedge \varphi_{\text{ft}} \wedge (\mu X.\varphi_{e_1} \wedge \neg\varphi_{e_2} \vee \langle 1\rangle\, X \vee \langle 2\rangle\, X)^{\text{FBT}} \tag{2}$$

which captures that $\varphi_{e_1} \wedge \neg\varphi_{e_2}$ need to be unsatisfiable at any location in the tree, for the containment to hold between e_1 and e_2.

It is important to note that formula (2) is alternation-free since our translations produce alternation-free formulae (fixpoints are properly nested), and the negation of an alternation free sentence remains alternation-free.

The practical experiments presented in the next section consider the positive normal form of (2), obtained by applying the rules given on Figure 4.

4. EXPERIMENTAL RESULTS

The objective of the section aims at testing the practical performance of our method. The proposed approach has been fully implemented. A compiler takes two XPath expressions as input, and translates them into $\mathcal{L}_\mu$ formulae. The containment formula is then composed, normalized and solved.

The satisfiability solver is based on the tableau method described in [34]. It is specialized for the alternation-free μ-calculus with converse. 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 converse [37]. Whenever the containment does not hold, the solver outputs a counter-example XML tree.

We carried out two testing scenarios[6]. 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. This 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 12, and results together with total running times of the decision procedure are summarized on Figure 13. Obtained results show that all tests are solved in less than 0.6 seconds. This suggests that XPath expressions used in real-world scenarios can be reasonably handled in practice.

The second test series consists in comparing expressions taken from research papers on the containment of XPath expressions. Figure 14 presents the expressions we collected. Some have been used to show that checking XPath containment in general may become very hard. In particular, the

[6]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/closed_auctions/closed_auction/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 12: XPath Queries Taken from the XPathmark Benchmark.

Relation	Time (s)	
	$\subseteq$	$\supseteq$
$Q_1 \not\subset Q_2$	0.10	0.32
$Q_1 \not\subset Q_3$	0.02	0.02
$Q_1 \not\subset Q_4$	0.04	0.02
$Q_1 \supset Q_5$	0.06	0.07
$Q_1 \not\subset Q_6$	0.04	0.03
$Q_1 \not\subset Q_7$	0.05	0.03
$Q_1 \supset Q_8$	0.07	0.11
$Q_1 \not\subset Q_9$	0.06	0.06
$Q_2 \subset Q_3$	0.22	0.05
$Q_2 \subset Q_4$	0.23	0.06
$Q_2 \not\subset Q_5$	0.37	0.11
$Q_2 \not\subset Q_6$	0.23	0.07
$Q_2 \not\subset Q_7$	0.29	0.06
$Q_2 \not\subset Q_8$	0.46	0.16
$Q_2 \not\subset Q_9$	0.54	0.12
$Q_3 \supset Q_4$	0.01	0.02
$Q_3 \not\subset Q_5$	0.03	0.04
$Q_3 \not\subset Q_6$	0.01	0.03

Relation	Time (s)	
	$\subseteq$	$\supseteq$
$Q_3 \not\subset Q_7$	0.02	0.02
$Q_3 \not\subset Q_8$	0.04	0.07
$Q_3 \not\subset Q_9$	0.03	0.05
$Q_4 \not\subset Q_5$	0.04	0.05
$Q_4 \not\subset Q_6$	0.01	0.02
$Q_4 \not\subset Q_7$	0.02	0.02
$Q_4 \not\subset Q_8$	0.04	0.07
$Q_4 \not\subset Q_9$	0.03	0.04
$Q_5 \not\subset Q_6$	0.04	0.04
$Q_5 \not\subset Q_7$	0.05	0.04
$Q_5 \equiv Q_8$	0.12	0.12
$Q_5 \not\subset Q_9$	0.08	0.08
$Q_6 \not\subset Q_7$	0.02	0.02
$Q_6 \not\subset Q_8$	0.05	0.07
$Q_6 \not\subset Q_9$	0.04	0.05
$Q_7 \not\subset Q_8$	0.06	0.08
$Q_7 \not\subset Q_9$	0.04	0.06
$Q_8 \not\subset Q_9$	0.11	0.10

Figure 13: 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.

tree pattern homomorphism technique [29], which is incomplete, return false negatives for some of them. Figure 15 shows corresponding results of our implementation.

The second test series gives comparable results. Both results are especially acceptable for static analysis purposes, where such operations are performed at compile-time.

No practical comparison with prior work is provided since no experimental results have been reported elsewhere. These measurements are thus of special interest since they shed light, for the first time, on the cost of checking the containment in practice.

5. RELATED WORK

Extensive research has been conducted on XPath query containment. Different fragments of the XPath language have been studied. Among them, a core XPath fragment is frequently used. This fragment isolates the "child" axis noted "/" (and included in all fragments), the "descendant" axis (often noted "//" in the literature[7]), branching "[ ]", and wildcard "*" as the most important features, and is denoted by $XP^{\{*,//,[]\}}$. Decidability of containment for $XP^{\{*,//,[]\}}$ can be obtained by a translation to datalog with recursion. While containment is undecidable for general dat-

[7]Actually $p_1//p_2$ is defined as the abbreviated notation for p_1/descendant-or-self::node()/p_2 by the XPath standard formal semantics [13].

Relation	Time (s)	
	$\subseteq$	$\supseteq$
$E_1 \subset E_2$	0.92	0.56
$E_3 \supset E_4$	0.11	0.23
$E_3 \supset E_5$	0.12	0.31
$E_4 \supset E_5$	0.32	0.52
$E_6 \subset E_7$	0.55	0.29
$E_8 \subset E_9$	0.05	0.05
$E_{10} \equiv E_{11}$	0.02	0.02

Figure 15: Results and Running Times.

alog with recursion, it has been shown using chase techniques, that the datalog fragment needed for $XP^{\{*,//,[]\}}$ has a decidable containment problem [39]. More specifically, containment for $XP^{\{*,//,[]\}}$ is coNP-complete [29]. The containment mapping technique relies on a polynomial time tree homomorphism algorithm, which gives a sufficient but not necessary condition for containment of $XP^{\{*,//,[]\}}$ in general.

If any of the three constructs "*", "//", or "[ ]" is dropped then query containment is in PTIME [29]. In particular, containment for $XP^{\{//,[]\}}$ is shown to be in PTIME in [3], and [39] noted that containment for $XP^{\{//,*\}}$ is in PTIME too.

Authors of [31] show that containment for $XP^{\{*,//,[],|\}}$, while coNP-complete for an infinite alphabet, is in PSPACE for a finite alphabet. They also show that containment for $XP^{\{//,|\}}$ is complete for PSPACE.

A summary of complexity results for various XPath fragments, classified with respect to complexity classes can be found in [32].

Characterizations of the expressive power of these languages in terms of both logics and tree patterns are given in [7]. This work also studies structural properties such as closure properties focusing on the ability to perform basic boolean operations while remaining in the same fragment.

A related problem concerns XPath containment in presence of constraints. [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. Additionally, the containment problem is shown to be in EXPTIME for the fragments $XP^{\{//,[]\}}$, $XP^{\{//,[],|\}}$, $XP^{\{//,|\}}$ in the presence of DTDs [40].

Compared to all these previous works, the XPath fragment we consider is far more complete and much more realistic. The connection between XPath and μ-calculus is advocated in [5], but it has not been developed yet.

6. CONCLUSION

In this paper, we proposed a new logical approach for XPath decision problems. XPath queries are translated into the μ-calculus. XPath containment is expressed in terms of a formula 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 calculus, namely the alternation free fragment of the modal μ-calculus with converse, as the appropriate logic for modeling XML data and XPath queries. As a valuable outcome, we show how an XPath expression can be translated into a μ-calculus formula.

Second, we take advantage of this translation to reduce the XPath containment problem to satisfiability in $\mathcal{L}_\mu$. We obtain an effective decision procedure for XPath containment. The considered XPath fragment includes union, intersection, path composition together with all forward and backward axes, branching, boolean connectives, wildcards, and negation. This fragment is far more complete than other fragments addressed in previous studies. The upper-bound complexity for the containment over this fragment is in EXPTIME.

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.

Eventually, an additional benefit 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 characterizing the $\mathcal{L}_\mu$ fragment needed for the different XPath fragments. Such a connection might be used for obtaining lower complexity bounds for XPath decision problems, by specifically tuning the μ-calculus satisfiability decision procedure. It could also be used for characterizing XPath evaluation, which is reduced to $\mathcal{L}_\mu$ model-checking by our translations. Another direction of future work consists in studying the scalability of our approach for XPath decision problems under type constraints such as DTDs or XML-Schemas. This requires to translate type constraints in $\mathcal{L}_\mu$, which we know feasible since AFMC equals WS2S which exactly captures regular tree types. If such a combination proves efficient, it would yield a unifying framework for the analysis of XML-based languages.

7. 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, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. *Theoretical Computer Science*, 336(1):3–31, 2005.

[8] 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.

[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. Draper, P. Fankhauser, M. Fernández, A. Malhotra, K. Rose, M. Rys, J. Siméon, and P. Wadler. XQuery 1.0 and XPath 2.0 formal semantics, W3C working draft, September 2005. http://www.w3.org/TR/2005/WD-xquery-semantics-20050915/.

[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 and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. In *PODS '02: Proceedings of the twenty-first ACM Symposium on Principles of Database Systems*, pages 17–28, New York, NY, USA, 2002. ACM Press.

[18] G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *LICS '02: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 189–202, Washington, DC, USA, 2002. IEEE Computer Society.

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

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

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

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] R. Mateescu. Local model-checking of modal mu-calculus on acyclic labeled transition systems. In *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 281–295, London, UK, 2002. Springer-Verlag.

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

[30] F. Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46, 2002.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

APPENDIX

We recall the denotational semantics of our XPath fragment. The evaluation of a query over an unranked tree returns a set of nodes. The formal semantics functions $\mathcal{S}_e$ and $\mathcal{S}_p$ define the set of nodes respectively returned by expressions and paths, starting from a context node x in the tree t:

$$
\begin{array}{lcl}
\mathcal{S}_e[\![\cdot]\!]^t_\cdot & : & \mathit{Expression} \to \mathit{Node} \to \mathrm{Set}(\mathit{Node}) \\
\mathcal{S}_e[\![/p]\!]^t_x & = & \mathcal{S}_p[\![p]\!]^t_{\mathit{root}()} \\
\mathcal{S}_e[\![p]\!]^t_x & = & \mathcal{S}_p[\![p]\!]^t_x \\
\mathcal{S}_e[\![e_1 \mid e_2]\!]^t_x & = & \mathcal{S}_e[\![e_1]\!]^t_x \cup \mathcal{S}_e[\![e_2]\!]^t_x \\
\mathcal{S}_e[\![e_1 \cap e_2]\!]^t_x & = & \mathcal{S}_e[\![e_1]\!]^t_x \cap \mathcal{S}_e[\![e_2]\!]^t_x
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{S}_p[\![\cdot]\!]^t_\cdot & : & \mathit{Path} \to \mathit{Node} \to \mathrm{Set}(\mathit{Node}) \\
\mathcal{S}_p[\![p_1/p_2]\!]^t_x & = & \{x_2 \mid x_1 \in \mathcal{S}_p[\![p_1]\!]^t_x \wedge x_2 \in \mathcal{S}_p[\![p_2]\!]^t_{x_1}\} \\
\mathcal{S}_p[\![p[q]]\!]^t_x & = & \{x_1 \mid x_1 \in \mathcal{S}_p[\![p]\!]^t_x \wedge \mathcal{S}_q[\![q]\!]^t_{x_1}\} \\
\mathcal{S}_p[\![\mathrm{a}::\sigma]\!]^t_x & = & \{x_1 \mid x_1 \in \mathcal{S}_a[\![\mathrm{a}]\!]^t_x \wedge \mathit{name}(x_1) = \sigma\} \\
\mathcal{S}_p[\![\mathrm{a}::*]\!]^t_x & = & \{x_1 \mid x_1 \in \mathcal{S}_a[\![\mathrm{a}]\!]^t_x\}
\end{array}
$$

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:

$$
\begin{array}{lcl}
\mathcal{S}_q[\![\cdot]\!]^t_\cdot & : & \mathit{Qualifier} \to \mathit{Node} \to \mathit{Boolean} \\
\mathcal{S}_q[\![q_1 \text{ and } q_2]\!]^t_x & = & \mathcal{S}_q[\![q_1]\!]^t_x \wedge \mathcal{S}_q[\![q_2]\!]^t_x \\
\mathcal{S}_q[\![q_1 \text{ or } q_2]\!]^t_x & = & \mathcal{S}_q[\![q_1]\!]^t_x \vee \mathcal{S}_q[\![q_2]\!]^t_x \\
\mathcal{S}_q[\![\text{not } q]\!]^t_x & = & \neg\, \mathcal{S}_q[\![q]\!]^t_x \\
\mathcal{S}_q[\![p]\!]^t_x & = & \mathcal{S}_p[\![p]\!]^t_x \neq \emptyset
\end{array}
$$

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

$$
\begin{array}{lcl}
\mathcal{S}_a[\![\cdot]\!]^t_\cdot & : & \mathit{Axis} \to \mathit{Node} \to \mathrm{Set}(\mathit{Node}) \\
\mathcal{S}_a[\![\mathrm{child}]\!]^t_x & = & \mathit{children}(x) \\
\mathcal{S}_a[\![\mathrm{parent}]\!]^t_x & = & \mathit{parent}(x) \\
\mathcal{S}_a[\![\mathrm{descendant}]\!]^t_x & = & \mathit{children}^+(x) \\
\mathcal{S}_a[\![\mathrm{ancestor}]\!]^t_x & = & \mathit{parent}^+(x) \\
\mathcal{S}_a[\![\mathrm{self}]\!]^t_x & = & \{x\} \\
\mathcal{S}_a[\![\mathrm{descendant\text{-}or\text{-}self}]\!]^t_x & = & \mathcal{S}_a[\![\mathrm{descendant}]\!]^t_x \cup \mathcal{S}_a[\![\mathrm{self}]\!]^t_x \\
\mathcal{S}_a[\![\mathrm{ancestor\text{-}or\text{-}self}]\!]^t_x & = & \mathcal{S}_a[\![\mathrm{ancestor}]\!]^t_x \cup \mathcal{S}_a[\![\mathrm{self}]\!]^t_x \\
\mathcal{S}_a[\![\mathrm{preceding}]\!]^t_x & = & \{y \mid y \ll x\} \setminus \mathcal{S}_a[\![\mathrm{ancestor}]\!]^t_x \\
\mathcal{S}_a[\![\mathrm{following}]\!]^t_x & = & \{y \mid x \ll y\} \setminus \mathcal{S}_a[\![\mathrm{descendant}]\!]^t_x \\
\mathcal{S}_a[\![\mathrm{following\text{-}sibling}]\!]^t_x & = & \{y \mid y \in \mathit{child}(\mathit{parent}(x)) \wedge x \ll y\} \\
\mathcal{S}_a[\![\mathrm{preceding\text{-}sibling}]\!]^t_x & = & \{y \mid y \in \mathit{child}(\mathit{parent}(x)) \wedge y \ll x\}
\end{array}
$$

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.