# An efficient translation from a modal $\mu$-calculus over finite trees with converse to tree automata

Louis Jachiet, Pierre Genevès, Nabil Layaïda

# An efficient translation from a modal $\mu$-calculus with converse to tree automata

**Louis Jachiet** and **Pierre Genevès** and **Nabil Layaïda**
Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble France

**Abstract**

We present a direct translation from a sub-logic of $\mu$-calculus to non-deterministic automata of finite binary trees. The logic is an alternation-free modal $\mu$-calculus, restricted to finite trees and where formulae are cycle-free. This logic is expressive enough to encode significant fragments of query languages (such as Regular XPath). We have implemented our translation. Our prototype effectively solves static analysis problems that were beyond reach, such as the XPath query containment problem with DTD constraints of significant size.

## Introduction

Tree automata are tightly connected to expressive logics. Capturing the language of models of a given logical formula using a tree automaton constructed from the formula has proved to be an essential technique to show decidability and complexity bounds for a variety of logics.

In practice, however, such an automaton construction is not necessarily feasible efficiently from a given logical language which is Monadic Second Order complete (MSO-complete). This is one of the main reasons why automata-based decision procedures did not have the same success in practice as they had on the theoretical side (Pan, Sattler, and Vardi 2006; Ünel and Toman 2007). Implementations of satisfiability-testing algorithms for expressive logics rarely rely on automata-based techniques. Notable exceptions include MONA (Klarlund and Møller 2001; Klarlund, Møller, and Schwartzbach 2001) for the weak monadic second-order logic of two successors (WS2S) (Thatcher and Wright 1968; Doner 1970) and MLSolver (Friedmann and Lange 2010) for the full $\mu$-calculus (without converse modalities) (Kozen 1983). In general, however, automata-based decision procedures implementations are often outperformed by alternative techniques such as tableau methods (Pan, Sattler, and Vardi 2006; Tanabe et al. 2005; Genevès et al. 2015). Such techniques try to avoid one of the main weaknesses of automata-based techniques: the explicit representation and construction of automata in intermediate steps of the decision procedure. Such intermediate steps often involve extremely large automata and make the decision procedure fail even if the final automaton is actually small.

Nevertheless, there are applications where such an explicit construction of a tree automaton is key and inevitable. One such application is the static analysis of queries in the presence of schemas. More specifically, solving the problem of query containment under schema constraints. In this context, building a tree automaton has proved to be useful in decreasing the overall combined complexity of the problem by an exponential in the size of the schema (Libkin and Sirangelo 2010). Another application is the static typing of functional programs manipulating trees. In such settings, type checking is often performed by so-called type inference, an operation that requires as input an explicit tree automaton. For example, the type-checkers of (Benzaken et al. 2013) need to produce a tree automaton from some logical language representing statements à la XPath (ten Cate, Litak, and Marx 2010). Finally, query evaluators seeking performance can benefit from such a translation by incorporating the automata construction directly in their compilers (Arroyuelo et al. 2015). While such a translation is known to be feasible in theory, an efficient implementation has only been conjectured to be feasible so far. This implementation challenge together with the aforementioned applications motivated the present work.

In this paper, we investigate the construction of a finite tree automaton from a formula of a specific tree logic $\mathcal{L}_\mu$ which is MSO-complete. This logic is an alternation-free and cycle-free fragment of the $\mu$-calculus with backward modalities, interpreted over finite trees. Thanks to its expressive power and succinctness, this logic has found many applications in particular for the static analysis of queries and programs that process semi-structured data such as XML (Genevès, Layaïda, and Schmitt 2007; Libkin and Sirangelo 2010; Calvanese et al. 2010). An efficient decision procedure for testing the satisfiability of this logic has been successfully designed and implemented by Genevès et al. (2015). However, the procedure relies on an inverse tableau method that (only) looks for a single satisfying model. In this paper, we propose a technique for effectively building a tree automaton representing the set of all satisfying models of a given formula. To reach that goal, we build the automaton in one pass, in a way that is as parsimonious as possible, in particular for building the transitions.

### Related Work.

Several works addressed the translation of formulae into tree automata or their satisfiability. The seminal work on

MONA (Klarlund and Møller 2001; Klarlund, Møller, and Schwartzbach 2001) developed implementation techniques for WS2S which is succinct, but whose satisfiability is non-elementary. We consider the equally expressive $\mathcal{L}_\mu$ logic introduced in (Genevès, Layaïda, and Schmitt 2007) whose satisfiability has time complexity $2^{O(n)}$.

More recently (Libkin and Sirangelo 2008) a translation was introduced from CondXPath to tree automata in $2^{O(n)}$. This work was further developed in (Libkin and Sirangelo 2010) and (Francis, David, and Libkin 2011). The same result was obtained by (Björklund, Gelade, and Martens 2010). However, CondXPath represents a strict subset of $\mu$-calculus (FO-complete: complete with respect to First-Order logic (Marx 2004)) and thus less expressive than the tree logic used here (which is MSO-complete). In addition, both translations produce unranked tree automata for which efficient implementations are notoriously lacking (none have been reported).

In (Calvanese et al. 2010), the authors presented a translation from $\mu$XPath formulas (a MSO-complete $\mu$-calculus variant) to non-deterministic tree automata (NDTA) of size $2^{O(n^2)}$ states where $n$ is the size of the formula in an intermediate language corresponding to the language $\mathcal{L}_\mu$ without the cycle-freeness restriction (the cycle-freeness restriction has no impact on expressivity but it is unknown whether it has an impact on succinctness). Their translation has no reported implementation. $\mathcal{L}_\mu$ (with the cycle-freeness restriction) has found a wide range of applications. For example, $\mathcal{L}_\mu$ is used for analyzing: XML programs that are imperative (Reichenbach et al. 2009) and functional (Xu 2013; Genevès and Gesbert 2015); updates (Benedikt and Cheney 2010); SPARQL queries (Chekol et al. 2012); stylesheets (Bosch, Genevès, and Layaïda 2015) and javascript code (Fard, Mesbah, and Wohlstadter 2015).

The closest contribution to ours, in spirit, remains the one proposed and implemented in (Klarlund, Møller, and Schwartzbach 2001), although their logic is very different: syntactically much more succinct but of hyperexponential complexity for satisfiability.

## Contributions.

Our contributions are twofold: **(I)** we present a new direct translation from $\mathcal{L}_\mu$ to tree automata, the size of the produced automaton is, at most, $2^{O(n)}$. This improves on the best known translations: a first line of work also had the $2^{O(n)}$ bound but they were limited to a FO logic; a second line dealt with an MSO-complete logic but produced automata of size $2^{O(n^2)}$. **(II)** Our new method allows our prototype to leverage semi-implicit representation techniques and to avoid the construction of inaccessible states. The result is a parsimonious implementation with which we solve concrete problems that were out of reach. Specifically, we provide the first implementation of such a translation and show that it effectively solves the static analysis of XPath queries under large real-world schemas such as XHTML. So far, such problem instances were beyond reach. We also show that our prototype performs well against the existing $\mathcal{L}_\mu$ solver on logical tasks such as query containment.

# Preliminaries : Logical Context

In this section we present the logic $\mathcal{L}_\mu$ introduced in (Genevès, Layaïda, and Schmitt 2007) and recall some of its known properties. This logic is a variant of $\mu$-calculus operating on labeled finite binary trees. The results on $\mathcal{L}_\mu$ transfer to finite unranked trees thanks to the well-known "first-child next-sibling" bijective mapping.

**Focused Trees.** A focused tree is a tree with the additional information of a node in this tree. This node is said focused on. To change the focus in a focused tree, we have four "programs": $\langle 1\rangle$,$\langle 2\rangle$,$\langle \bar{1}\rangle$,$\langle \bar{2}\rangle$ (also used in the formulae $\langle a\rangle\,\varphi$ with $a \in \{1,2,\bar{1},\bar{2}\}$). For $\mathcal{T}$ a focused tree and $a \in \{1,2\}$, $\mathcal{T}\langle a\rangle$ (resp. $\mathcal{T}\langle \bar{a}\rangle$) denotes the same tree but where the focus is moved on the $a$-child (resp. the parent with $(\mathcal{T}\langle \bar{a}\rangle)\langle a\rangle = \mathcal{T}$). Obviously, $\mathcal{T}\langle a\rangle$ (resp. $\mathcal{T}\langle \bar{a}\rangle$) is only defined if the node we are focused on has a $a$-child (resp. if it is the $a$-child of some node). We write $\mathcal{F}$ for the set of all focused trees.

We suppose an alphabet $\Sigma$ and that each node of a tree we consider is labeled by $l \subseteq \Sigma$. Given a focused tree $\mathcal{T}$ we note $\mathcal{L}abel(\mathcal{T})$ the label of the node focused by $\mathcal{T}$.

**Syntax of formulae.** The logic we consider is an alternation-free cycle-free modal $\mu$-calculus presented below. We suppose, to simplify things, that each variable name is only bound once (eventually thanks to an $\alpha$-conversion). The syntax is given in figure 1.

**Atomic propositions.** For each $P \in \Sigma$, the atomic proposition $P$ is satisfied by $\mathcal{T}$ when $P \in \mathcal{L}abel(\mathcal{T})$.

**Existential Modality.** Existential modalities are formulae of the form $\langle a\rangle\,\varphi$ for $a \in \{1,2,\bar{1},\bar{2}\}$. $\langle a\rangle\,\varphi$ is satisfied by $\mathcal{T}$ when $\mathcal{T}\langle a\rangle$ exists and satisfies $\varphi$.

**Negated formulae.** Only atomic propositions, existential modalities of the form $\langle a\rangle\top$, and $\top$ can be negated ($\top$ has its usual "true" meaning).

**Fixpoints.** The formulae of the form $\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi$ correspond to fixpoints. Each fixpoint has its variables indexed by a finite set $I$ (each fixpoint can have a different $I$). $\mathcal{L}_\mu$ was introduced with a least n-ary fixpoint ($\mu$) and a greatest n-ary fixpoint ($\nu$) but both fixpoints have the same interpretation for cycle-free formulae (Genevès, Layaïda, and Schmitt 2007).

**Interpretation of formulae.** The set of models of a formula $\varphi$ (i.e. the focused trees satisfying $\varphi$) is recursively defined as $[\![\varphi]\!]_V$ where $V$ is the environment mapping the

| | | | |
|---|---|---|---|
| $\varphi$ | $::=$ | $\varphi_1 \wedge \varphi_2$ | conjunction |
| | $\mid$ | $\varphi_1 \vee \varphi_2$ | disjunction |
| | $\mid$ | $P \mid \neg P$ | atomic propositions (negated) |
| | $\mid$ | $\top \mid \neg\top$ | true, false |
| | $\mid$ | $\langle a\rangle\,\varphi \mid \neg\langle a\rangle\top$ | existential modality (negated) |
| | $\mid$ | $\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi$ | polyadic fixpoint |
| | $\mid$ | $X$ | variable |

Figure 1: Syntax of formulae.

$$
\begin{array}{rcl}
[\![P]\!]_V & = & \{\,\mathcal{T} \in \mathcal{F} \mid P \in \mathcal{L}abel(\mathcal{T})\} \\
[\![\top]\!]_V & = & \mathcal{F} \\
[\![\langle a\rangle\,\varphi]\!]_V & = & \{\mathcal{T} \in \mathcal{F} \mid (\exists\, \mathcal{T}\,\langle a\rangle) \\
 & & \wedge \mathcal{T}\,\langle a\rangle \in [\![\varphi]\!]_V\} \\
[\![X]\!]_V & = & V(X) \\
[\![\varphi_1 \wedge \varphi_2]\!]_V & = & [\![\varphi_1]\!]_V \cap [\![\varphi_2]\!]_V \\
[\![\varphi_1 \vee \varphi_2]\!]_V & = & [\![\varphi_1]\!]_V \cup [\![\varphi_2]\!]_V \\
[\![\neg\varphi]\!]_V & = & \mathcal{F} \setminus [\![\varphi]\!]_V \\
[\![\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi]\!]_V & = & [\![\psi]\!]_{V[X_i \to U_i]_{i\in I}}
\end{array}
$$

$$\text{With } U_j = \bigcap_{(T_i)_{i\in I}\in S} T_j \text{ and}$$
$$S = \{(T_i)_{i\in I} \in \mathcal{P}(\mathcal{F})^I \mid \forall j([\![\varphi_j]\!]_{V[X_i\to T_i]_{i\in I}} \subseteq T_j)\}.$$

Figure 2: Interpretation of formulae.

free variables of $\varphi$ to sets of models. Figure 2 presents the definition of $[\![\varphi]\!]_V$. The notation $V[A \to B]$ indicates that $V$ is modified to add a binding from the variable $A$ to the set of models $B$ (with the unicity of variables names we never replace a binding). For closed formulae we omit the environment (i.e. we write $[\![\varphi]\!]$).

## Syntactic graph.

**Definition 1.** *The expansion of a fixpoint formula is defined as: exp*$(\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi) = \psi\{X_j/\mu(X_i = \varphi_i)_{i\in I}\ in\ \varphi_j\}$ *where* $\psi\{X_j/\mu(X_i = \varphi_i)_{i\in I}\ in\ \varphi_j\}$ *is the formula* $\psi$ *where occurrences of the variable* $X_j$ *(for* $j \in I$*) are replaced by* $\mu(X_i = \varphi_i)_{i\in I}\ in\ \varphi_j$.

Intuitively fixpoints are defined by recursive formulae and the expansion operation "unfolds" the first step of this recursive operation.

**Definition 2.** *The syntactic graph is the graph whose vertices are formulae and edges link formulae with their direct subformulae. All edges are labeled by a program or the absence thereof (noted* $\epsilon$*) denoting the modality crossed. Edges linking a formula* $\langle a\rangle\,\varphi$ *with* $\varphi$ *are labeled by* $\langle a\rangle$ *and all other edges are labeled by* $\epsilon$*. Formally the edges are:*

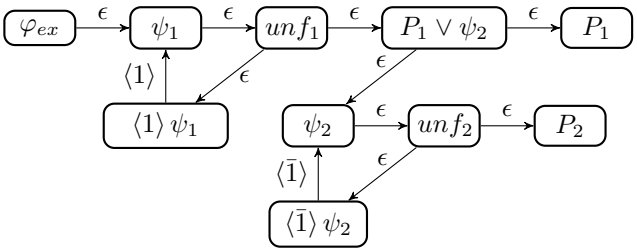
- $\varphi_1 \wedge \varphi_2 \xrightarrow{\epsilon} \varphi_i$, $\varphi_1 \vee \varphi_2 \xrightarrow{\epsilon} \varphi_i$ *for* $i \in \{1,2\}$; $\neg\varphi \xrightarrow{\epsilon} \varphi$;
- $\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi \xrightarrow{\epsilon} exp(\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi)$;
- $\langle a\rangle\,\varphi \xrightarrow{\langle a\rangle} \varphi$ *for* $a \in \{1, 2, \bar{1}, \bar{2}\}$.

**Definition 3.** *The* Fischer-Ladner *closure* $cl(\theta)$ *of a closed formula* $\theta$ *is the set of formulae reachable from* $\theta$ *in the syntactic graph.*

Let us recall (see (Kozen 1983) or (Genevès et al. 2015)) propositions on $\mathcal{L}_\mu$:

**Proposition 1.** *Expanding a fixpoint does not change its semantics, i.e.* $[\![\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi]\!]_V = [\![exp(\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi)]\!]_V$

**Proposition 2.** *The* Ficher-Ladner *closure of a closed formula* $\theta$ *is finite and its size is linear in the size of* $\theta$.




$I = \{1, 2\}$ $\qquad$ $\psi_j = \mu(X_i = \varphi_i)_{i\in I}\ in\ \varphi_j$
$\varphi_1 = (P_1 \vee X_2) \wedge \langle 1\rangle\, X_1$ $\qquad$ $\varphi_2 = P_2 \vee \langle \bar{1}\rangle\, X_2$
$unf_1 = (P_1 \vee \psi_2) \wedge \langle 1\rangle\,\psi_1$ $\qquad$ $unf_2 = P_2 \vee \langle \bar{1}\rangle\,\psi_2$
(note $unf_j = \exp(\psi_j)$) $\qquad$ $\varphi_{ex} = \mu(X_i = \varphi_i)_{i\in I}\ in\ X_1$

Figure 3: Example of Fischer-Ladner closure

**Types.**

**Definition 4.** *Types are similar to hintikka sets in the temporal logic literature. Types are elements of* $2^{cl(\theta)}$ *that are "downward-saturated", formally: the set* $\mathcal{T}ypes(\theta)$ *of* $\theta$*-types is the subset of* $2^{cl(\theta)}$ *such that* $\mathbf{t} \in \mathcal{T}ypes(\theta)$ *when for each* $\psi \in cl(\theta)$ *we have:*

- $\psi = \top \Rightarrow (\top \in \mathbf{t})$
- $\psi = \varphi_1 \wedge \varphi_2 \Rightarrow \Big((\psi \in \mathbf{t}) \Leftrightarrow (\varphi_1 \in \mathbf{t} \wedge \varphi_2 \in \mathbf{t})\Big)$
- $\psi = \varphi_1 \vee \varphi_2 \Rightarrow \Big((\psi \in \mathbf{t}) \Leftrightarrow (\varphi_1 \in \mathbf{t} \vee \varphi_2 \in \mathbf{t})\Big)$
- $\psi = \neg\varphi \Rightarrow \Big((\psi \in \mathbf{t}) \Leftrightarrow (\varphi \notin \mathbf{t})\Big)$
- $\psi = \mu(X_i = \varphi_i)_{i\in I}\ in\ \tau \Rightarrow \Big(\psi \in \mathbf{t} \Leftrightarrow exp(\psi) \in \mathbf{t})\Big)$

## Cycle-freeness.

It is possible in $\mathcal{L}_\mu$ to write recursive formulae (i.e. fixpoints) but such recursions have to be "cycle-free". We present here the distinction between non "cycle-free" formulae where the focus can come back infinitely many times to the same node (i.e. where the recursion "loops" e.g. $\mu X = X\ in\ X$ or $\mu X = \langle 1\rangle\, X \vee \langle \bar{1}\rangle\, X\ in\ X$) and "cycle-free" formulae such as $\varphi_{ex}$ (see figure 3) where the recursion "terminates". Note that in cycle-free formulae the focus can come back to the same node but each node in a tree is visited a finite number of times. For instance, in $\varphi_{ex}$ the focus starts on a node, repeatedly (an unbounded number of times) moves down to its 1-child then repeatedly moves to its 1-parent, each node is thus visited, at most, twice.

Given a walk $v_1 \xrightarrow{l_1} v_2 \ldots \xrightarrow{l_n} v_{n+1}$ in the syntactic graph, the corresponding *path* is the sequence of labels $l_1 \ldots l_n$. The set of paths for a formula $\theta$ is the set of paths corresponding to walks starting from $\theta$. For $\varphi_{ex}$ (fig. 3), paths are prefixes of $\{\epsilon^3(\langle 1\rangle\,\epsilon^2)^n\epsilon^3(\langle \bar{1}\rangle\,\epsilon^2)^m | (n,m) \in \mathbb{N}^2\}$.

Finally, a path $p_1 \ldots p_n$ is *valid* on a focused tree $\mathcal{T}$ if for $1 \leq i \leq n$ all the $\mathcal{T}p_1 \ldots p_i$ exist (we use the convention $\mathcal{T}\epsilon = \mathcal{T}$).

**Definition 5.** *A formula* $\varphi$ *is* cycle-free *if for every focused tree* $\mathcal{T}$ *there is an integer* $c(\varphi, \mathcal{T})$ *such that no path of* $\varphi$ *longer than* $c(\varphi, \mathcal{T})$ *is a valid path of* $\mathcal{T}$.

From now on, we consider a closed $\mathcal{L}_\mu$ formula $\theta$, we want to produce a tree automaton that recognizes a tree $T$ if the focused tree $\mathcal{T}$ focused on the root of $T$ is a model of $\theta$ (i.e. $\mathcal{T} \in [\![\theta]\!]$). From the set $cl(\theta)$ we extract the following subsets:

- $\Sigma = \{P \in cl(\theta) \mid P \text{ is an atomic proposition}\}$;
- $F_a = \{\langle a \rangle \varphi \mid \langle a \rangle \varphi \in cl(\theta)\}$ for $(a \in \{1, 2, \bar{1}, \bar{2}\})$;
- $\mathcal{L}\text{ean} = \Sigma \cup_{a \in \{1,2,\bar{1},\bar{2}\}} F_a$.

## Annotation of Trees

In this section, we introduce the notions of annotation and two properties on them: local consistency and consistency. Then, we prove that those two properties are actually equivalent. Local consistency will be used to derive an automaton, while consistency helps in establishing that the automaton captures the set of trees that are models of the formula.

**Definition 6.** *A type* $\mathbf{t}$ *is* consistent *with a focused tree* $\mathcal{T}$ *when* $\forall \varphi \in cl(\theta)\ (\varphi \in \mathbf{t} \Leftrightarrow \mathcal{T} \in [\![\varphi]\!])$.

**Definition 7.** *A* annotation $\gamma$ *of a finite tree is a function from the nodes to types.* $\gamma$ *is* consistent *when each node* $\mathcal{N}$ *is associated with a type that is consistent with the tree focused on* $\mathcal{N}$*. Given an annotation* $\gamma$ *and a focused tree* $\mathcal{T}$*,* $\gamma(\mathcal{T})$ *refers to the type associated with the node focused by* $\mathcal{T}$*.*

**Definition 8.** *Given two types* $\mathbf{t}$*,* $\mathbf{t}_a$ *and* $a \in \{1, 2\}$*, the* $\Delta_a$*-compatibility is (with* $\dot{1} = \bar{2}$ *and* $\dot{2} = \bar{1}$*):*

$$\Delta_a(\mathbf{t}, \mathbf{t}_a) = \begin{cases} \forall \langle a \rangle \varphi \in F_a & \langle a \rangle \varphi \in \mathbf{t} \Leftrightarrow \varphi \in \mathbf{t}_a \\ \forall \langle \bar{a} \rangle \varphi \in F_{\bar{a}} & \langle \bar{a} \rangle \varphi \in \mathbf{t}_a \Leftrightarrow \varphi \in \mathbf{t} \\ \forall \langle \dot{a} \rangle \varphi \in F_{\dot{a}} & \langle \dot{a} \rangle \varphi \notin \mathbf{t}_a \end{cases}$$

**Definition 9.** *We say that an annotation* $\gamma$ *of the tree* $T$ *is locally consistent when for each node* $\mathcal{N}$ *of* $T$*:*

- $\forall P \in \Sigma\ (P \in \gamma(\mathcal{N}) \Leftrightarrow P \in \mathcal{L}abel(\mathcal{N}))$*;*
- *for* $a \in \{1, 2\}$
  - *if* $\mathcal{N}$ $\langle a \rangle$ *exists then* $\Delta_a(\gamma(\mathcal{N}), \gamma(\mathcal{N}\langle a \rangle))$*;*
  - *if* $\mathcal{N}$ *has no* $a$*-child then* $F_a \cap \gamma(\mathcal{N}) = \emptyset$*;*
  - *if* $\mathcal{N}$ *is the root then* $(F_{\bar{1}} \cup F_{\bar{2}}) \cap \gamma(\mathcal{N}) = \emptyset$*.*

**Theorem 1** (Local consistency is consistency)**.** *An annotation is locally consistent if and only if it is consistent.*

*Proof Sketch.* The idea of the proof (see appendix) relies on a notion of consistency at range $k$. The consistency at range 0 is defined as the local consistency.

Given a focused tree $\mathcal{T}$ and $k > c(\theta, \mathcal{T})$ (where $c(\theta, \mathcal{T})$ was introduced in the cycle-freeness criterion) we show that the consistency at range $k$ is equivalent to the consistency.

Finally, we show that the consistency at range $k$ is equivalent to the consistency at range $k + 1$ which proves (by recursion) that the local consistency is equivalent to the consistency. □

$$\begin{array}{lcl} \mathcal{Q} & = & \{\#\} \cup \mathcal{T}ypes(\theta) \\ \mathcal{E} & = & \{\mathbf{t} \in \mathcal{T}ypes(\theta) \mid \theta \in \mathbf{t} \wedge (F_{\bar{1}} \cup F_{\bar{2}}) \cap \mathbf{t} = \emptyset\} \\ \mathcal{A} & = & 2^{\Sigma} \\ \nu & = & \{(s_1, s_2, \mathbf{t}, \mathbf{t} \cap \Sigma) \mid \mathbf{t} \in \mathcal{T}ypes(\theta) \text{ and for } a \in \{1, 2\} \\ & & \text{either } s_a = \# \text{ and } \mathbf{t} \cap F_a = \emptyset \text{ or } \Delta_a(\mathbf{t}, s_a)\} \end{array}$$

Figure 4: Automaton $B_\theta$ based on local consistency

## Automaton construction

### Non-Deterministic Tree Automata.

The automata we consider in this paper are finite Non-Deterministic Tree Automata (NDTA) of the form $(\mathcal{A}, \mathcal{Q}, \delta, \mathcal{E}, \#)$. $\mathcal{A}$ is the alphabet of the automata, $\mathcal{Q}$ is the set of states, $\delta \subseteq \mathcal{Q}^3 \times \mathcal{A}$ is the set of transitions, $\mathcal{E}$ is the set of final states and $\# \in \mathcal{Q}$ is the initial state. The sets $\mathcal{A}$, $\mathcal{Q}$, $\delta$ and $\mathcal{E}$ are all finite.

A run of a NDTA against the finite binary tree $T$ is a function from the nodes of $T$ to $\mathcal{Q}$ such that each node $n$ labelled $l$ is associated with a state $q$ such that $(q_1, q_2, q, l) \in \delta$, where $q_i$ is the state associated with the $i$-child of $n$ if it exists or $\#$ if it doesn't exist.

A tree $T$ is accepted if there is a run that associates the root node with a state of $\mathcal{E}$.

### An Automaton for the Local Consistency.

As a consequence of our theorem, we can easily produce a NDTA that accepts the models of a formula $\theta$. The automaton $B_\theta = (\mathcal{A}, \mathcal{T}, \nu, \mathcal{E}, \#)$ (see figure 4) uses types as the automaton states; the transitions enforce the local consistency; the final states checks the root condition for local consistency and that the tree focused on the root satisfies $\theta$.

**Size of** $B_\theta$ A careful analysis of types show that there are $2^{|\mathcal{L}\text{ean}|}$ types. This means our automaton has $2^{|\mathcal{L}\text{ean}|}$ states and, at most, $2^{3\times|\mathcal{L}\text{ean}|}$ transitions. Since the size of $cl(\theta)$ is linear in the size of $\theta$ and the $\mathcal{L}\text{ean} \subseteq cl(\theta)$, we have the exponential bound.

### A Better Automaton.

$B_\theta$ always has $2^{|\mathcal{L}\text{ean}|}$ states, we now present a second automaton, $C_\theta = (\mathcal{A}, \mathcal{Q}', \delta', \mathcal{F}, \#)$ (see figure 5) that is also exponential in the size of the formula but is much more implementation friendly with less states in the worst case and even less states in the average case.

$B_\theta$ associates nodes of the tree with types, $C_\theta$ associates nodes with sets of candidate types for the parent node. We note $\mathcal{P}_a(q)$ the function that associates a state $q$ and a side $a \in \{1, 2\}$ to this set of potential $a$-parent types.

States of $C_\theta$ are: $\top$ the unique final state; $\#$ the leaf state and $(\mathcal{S}_i(\mathbf{t}), i)$ where $\mathcal{S}_i(\mathbf{t})$ is the set of types compatible with being the $i$-parent of $\mathbf{t}$. Each transition $(q_1, q_2, q, l)$ of $C_\theta$ is built using a type $\mathbf{t}$: $\mathbf{t}$ is a compatible $a$-parent for both $q_a$ ($\mathbf{t} \in \mathcal{P}_a(q_a)$), $\mathbf{t}$ is compatible with $l$ ($\mathbf{t} \cap \Sigma = l$) and either $q = \top$ (when $\mathbf{t} \in \mathcal{E}$) or $q = (\mathcal{S}_i(\mathbf{t}), i)$.

This construction works because there is a bijection between accepting runs of $C_\theta$ and accepting runs of $B_\theta$. It is

$$\mathcal{P}_i(\top) = \emptyset \qquad \mathcal{P}_i((e,a)) = \emptyset \text{ when } a \neq i$$
$$\mathcal{P}_i((e,a)) = e \text{ when } a = i \qquad \mathcal{P}_i(\#) = \#_i$$

where $\#_i = \{\mathbf{t} \in \mathcal{T}ypes(\theta) \mid \mathbf{t} \cap F_i = \emptyset\}$ is the set of types compatible with having no $i$-children.

$$\begin{aligned}
\mathcal{S}_i(\mathbf{t}) &= \{\mathbf{t}' \in \mathcal{T}ypes(\theta) \mid \Delta_i(\mathbf{t}', \mathbf{t})\} \\
\mathcal{Q}' &= \{\#, \top\} \cup \big\{(\mathcal{S}_i(\mathbf{t}), i) \mid (\mathbf{t}, i) \in \mathcal{T}ypes(\theta) \times \{1,2\}\big\} \\
\mathcal{F} &= \{\top\} \\
\delta &= \big\{(q_1, q_2, (\mathcal{S}_i(\mathbf{t}), i), \mathbf{t} \cap \Sigma) \;\big|\; i \in \{1,2\} \wedge \\
&\qquad \mathbf{t} \in \big(\mathcal{P}_1(q_1) \cap \mathcal{P}_2(q_2)\big)\big\} \;\bigcup \\
&\qquad \big\{(q_1, q_2, \top, \mathbf{t} \cap \Sigma) \;\big|\; \mathbf{t} \in \mathcal{P}_1(q_1) \cap \mathcal{P}_2(q_2) \cap \mathcal{E}\big\}
\end{aligned}$$

Figure 5: Automaton $C_\theta = (\mathcal{A}, \mathcal{Q}', \delta', \mathcal{F}, \#)$

easy to transform an accepting run $\gamma$ of $B_\theta$ into an accepting run $\eta$ of $C_\theta$: $\eta(n) = (S_i(\gamma(n)), i)$ when $n$ is a $i$-child and $\eta(r) = \top$ where $r$ is the root.

Let us show how to transform an accepting run $\eta$ of $C_\theta$ into a locally consistent annotation $\gamma$. For any non-root node $n$, $n$ is the $i$-child of some node and thus there is $\mathbf{t}$ such that $\eta(n) = (\mathcal{S}_i(\mathbf{t}), i)$ and $\eta(n)$ was obtained using a transition of the form $(q_1, q_2, \eta(n), \mathcal{L}abel(n))$ with $\mathbf{t} \in \mathcal{P}_1(q_1) \cap \mathcal{P}_2(q_2)$, $\mathcal{L}abel(n) = \mathbf{t} \cap \Sigma$ and $q_a$ the state of the $a$-child (or $\#$). Let $\gamma(n)$ be such a $\mathbf{t}$. The run is accepting, so the root node $r$ is associated with $\eta(r) = \top$. $\top$ comes from a transition of the form $(q_1, q_2, \mathcal{L}abel(r), \top)$ and that ensures there is a $\mathbf{t}$ in $\mathcal{P}_1(q_1) \cap \mathcal{P}_2(q_2) \cap \mathcal{E}$ with $\mathbf{t} \cap \Sigma = \mathcal{L}abel(r)$; let $\gamma(r)$ be such a $\mathbf{t}$. Let us show that $\gamma$ *is locally consistent:*

- a node $n$ is labelled with $\gamma(n) \cap \Sigma$;
- let $n_i$ be the $i$-child of $n$, we have $\gamma(n) \in \mathcal{P}_i(\eta(n_i)) = \mathcal{S}_i(\gamma(n_i))$, thus $\Delta_i(\gamma(n), \gamma(n_i))$;
- let $n$ be a node with no $i$-child, $\gamma(n) \in \#_i$ which implies $F_i \cap \gamma(n) = \emptyset$;
- let $r$ be the root, $\gamma(r) \in \mathcal{E}$ thus $\gamma(r) \cap (F_{\bar{1}} \cup F_{\bar{2}}) = \emptyset$.

Finally this locally consistent annotation associates the root with $\gamma(r) \in \mathcal{E}$, therefore $\gamma$ is an accepting run of $B_\theta$.

### Size of $C_\theta$

In this second automaton, states are not types but sets of types. There are much more sets of types than types but we only consider some of the sets of types and the resulting automaton $C_\theta$ is much smaller than the first automaton $B_\theta$.

A set $\mathcal{S}_i(\mathbf{t})$ is characterized by which formulae $\varphi$ are true on $\mathbf{t}$ (for $\langle i \rangle \varphi \in F_i$ or $\varphi \in F_{\bar{i}}$). Therefore, the number of distinct $\mathcal{S}_i$ sets is bounded by $2^{|F_i \cup F_{\bar{i}}|}$. The number of states of our automaton is bounded by the number of distinct sets $\mathcal{S}_1$ plus the number of distinct sets $\mathcal{S}_2$ plus two ($\#$ and $\top$). We have $2 + 2^{|F_1 \cup F_{\bar{1}}|} + 2^{|F_2 \cup F_{\bar{2}}|} = 2^{O(n)}$ states.

**Lemma 1.** *Given two types* $\mathbf{x}$ *and* $\mathbf{y}$ *and* $a \in \{1,2\}$ *we either have* $S_a(\mathbf{x}) \cap S_a(\mathbf{y}) = \emptyset$ *or* $S_a(\mathbf{x}) = S_a(\mathbf{y})$.

*Proof.* Let $\mathbf{z} \in S_a(\mathbf{x}) \cap S_a(\mathbf{y})$ and $\mathbf{w} \in S_a(\mathbf{x})$. We have: $\forall \langle a \rangle \varphi \in F_a \quad (\varphi \in \mathbf{y}) = (\langle a \rangle \varphi \in \mathbf{z}) = (\varphi \in \mathbf{x}) = (\langle a \rangle \varphi \in \mathbf{w})$ ; $\forall \langle \bar{a} \rangle \varphi \in F_{\bar{a}} \quad (\langle \bar{a} \rangle \varphi \in \mathbf{y}) = (\varphi \in \mathbf{z}) = (\langle \bar{a} \rangle \varphi \in \mathbf{x}) = (\varphi \in \mathbf{w})$ and $\mathbf{w} \in S_a(\mathbf{x}) \Rightarrow \forall \langle \dot{a} \rangle \varphi \in F_{\dot{a}}\ \langle \dot{a} \rangle \varphi \notin \mathbf{w}$, thus $\Delta_a(\mathbf{w}, \mathbf{y})$ and $\mathbf{w} \in S_a(\mathbf{y})$. □

Each transition $(q_1, q_2, q, l)$ is built using a type $\mathbf{t} \in \mathcal{P}_1(q_1) \cap \mathcal{P}_2(q_2)$ with $l = \mathbf{t} \cap \Sigma$. As a consequence of lemma 1 for $a \in \{1,2\}$, $\mathbf{t}$ can only belong to one $S_a(\mathbf{t}')$ (there might be several $\mathbf{t}''$ such that $S_a(\mathbf{t}') = S_a(\mathbf{t}'')$ but they all correspond to the same state $\big(S_a(\mathbf{t}'), a\big)$). Depending on whether $\mathbf{t} \in \#_a$ and the existence of $\mathbf{t}_a$ such that $\mathbf{t} \in \mathcal{S}_a(\mathbf{t}_a)$ there are, at most, two possibilities for each $q_a$ ($q_a = \#$ or $q_a = (\mathcal{S}_a(\mathbf{t}_a), i)$) Depending on whether $\mathbf{t}$ is compatible with a 1-parent, a 2-parent, or a root solution, there are, at most, three possible states for $q$: $\big(S_1(\mathbf{t}), 1\big)$, $\big(S_2(\mathbf{t}), 2\big)$ and $\top$. The automaton has thus, at most, $2 \times 2 \times 3 \times \mathcal{T}ypes(\theta) = 12 \times 2^{\mathcal{L}ean} = 2^{O(n)}$ transitions.

### Algorithm Implementation.

Given a pair of states $(q_1, q_2)$, we can compute the transitions using those states as children (i.e. of the form $(q_1, q_2, q, l)$) by iterating through $\mathcal{P}_1(q_1) \cap \mathcal{P}_2(q_2)$. In order to compute it efficiently, our prototype relies on a semi-implicit representation of sets of types for fast enumerations.

To compute the automaton, we gradually compute the list of accessible states. We start with only $\#$ marked accessible, then for each pair $(q_1, q_2)$ of accessible states we compute all the transitions $(q_1, q_2, q, l)$ and mark the $q$ as accessible.

## Experimental Validation

The source code of the prototype and the benchmark are available at the address: `http://place-holder/`.

Tree automata are needed for a variety of XPath-related applications: typechecking (Benzaken et al. 2013), query evaluation (Arroyuelo et al. 2015) and a variety of static analysis problems involving XPath queries (Schwentick 2007). Our first set of experiments aims at assessing the relevance of our method in this context. We translate XPath queries to tree automata and show that our translation can be done in reasonable time for "real-world" queries and that the resulting automaton is relatively small (much less than the $O(2^n)$ worst case), enabling further analyses on these automata.

### Typing XPath.

The XPathMark (Franceschet 2005) is a set of queries introduced to benchmark the major aspects of the XPath language. Our benchmark is a subset of XPathMark queries. We kept only the queries that can be translated into $\mathcal{L}_\mu$ without approximation. We did not consider the set of "$C_i$" queries that contain comparisons between data values: a feature that makes static analysis tasks such as query containment undecidable. XPathMark queries $B_{10}$, $B_{11(i)}$, $B_{12(i)}$, $B_{13(i)}$, $B_{14(i)}$, $B_{15(i)}$ for $i = 3$ are translated into tree automata in: 48.24 0.09, 0.18, 0.05, 0.05 and 61.44 seconds, respectively. These queries translate into either empty or very small automata, because their structure is simple.

Figure 6 presents for each query the time spent (in seconds) to compute $C_\theta$, the $\lg_2$ ($log$ in base 2) of: the number

| Query | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **A7** | **A8** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** | **B8** | **B9** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time (s) | 0.30 | 0.00 | 0.02 | 2.14 | 0.02 | 0.96 | 0.06 | 0.67 | 0.52 | 0.01 | 0.02 | 0.02 | 0.35 | 0.66 | 0.06 | 0.12 | 52.58 |
| $\lg_2(\#\text{states } B_\theta)$ | 22 | 13 | 17 | 25 | 18 | 23 | 19 | 23 | 23 | 16 | 17 | 17 | 24 | 23 | 17 | 22 | 42 |
| $\lg_2(\#\text{states } C_\theta)$ | 8.1 | 5.4 | 6.5 | 9.4 | 6.3 | 9.2 | 7.2 | 9.9 | 8.2 | 5.9 | 6.3 | 6.3 | 8.9 | 9.2 | 6.4 | 7.6 | 13.9 |
| $\lg_2(\#\text{trans } C_\theta)$ | 10.1 | 7.4 | 8.9 | 15.5 | 10.0 | 14.6 | 11.4 | 14.3 | 11.5 | 9.1 | 9.3 | 9.3 | 13.3 | 11.9 | 10.8 | 11.3 | 14.2 |

Figure 6: Statistics of the translation of the XPathMark Benchmark.

| Problem | Answer | inter-$a4\mu$ | full-$a4\mu$ | btl |
|---|---|---|---|---|
| $e1 \subseteq e2$ | $Yes$ | **0.84** | 2.20 | 2.82 |
| $e2 \subseteq e1$ | $No$ | **1.00** | 2.26 | 2.67 |
| $e3 \subseteq e4$ | $Yes$ | **0.06** | 0.52 | 0.97 |
| $e4 \subseteq e3$ | $Yes$ | **0.05** | 0.48 | 1.15 |
| $e5 \subseteq e6$ | $No$ | 7.42 | 610.54 | **0.85** |
| $e6 \subseteq e5$ | $Yes$ | **6.81** | 596.53 | 8.88 |

| Problem | $DTD$ | Answer | inter-$a4\mu$ | btl |
|---|---|---|---|---|
| $e8$ | $None$ | $Sat$ | **0.01** | 0.19 |
| $e8$ | XHTML | $Sat$ | **0.11** | 2.26 |
| $e9 \subseteq e13$ | $None$ | $No$ | **0.01** | 0.23 |
| $e9 \subseteq e13$ | XHTML | $Yes$ | **0.15** | 3.5 |

Figure 7: Time (in seconds) spent by the $a4\mu$ and the btl-solver on various satisfiability problems.

of states of $B_\theta$ (i.e. $2^{|\Sigma ean|}$), number of states of $C_\theta$, and the number of transitions of $C_\theta$. For most queries, the translation is done in less than a second and the size of $C_\theta$ is much smaller than the worst case (in the worst case $C_\theta$ can have as many transitions as the number of $B_\theta$ states).

### The Query Containment Problem.

Testing the containment $e \subseteq e'$ *(are all models of* $e$ *also models of* $e'$*?)* can be done by checking the satisfiability of $e \wedge \neg e'$ or by intersecting the automata for $\wedge e'$ and $e$.

To benchmark our prototype we also used the containment problems ($e1$ to $e12$ with $e13 = e10 \cup e11 \cup e12$) from the paper that introduced the btl-solver (Genevès et al. 2015). Figure 7 presents the results of our benchmark comparing three methods: translating both formulae to automata and intersecting them *(method inter-*$a4\mu$*)*, translating directly the formula $e \wedge \neg e'$ to automata and testing its emptyness *(method full-*$a4\mu$*)* and testing the satisfiability with the btl-solver.

Unsurprisingly the method *full-*$a4\mu$ is outperformed by the methods *inter-*$a4\mu$ and *btl-solver*. The mixed comparison between *inter-*$a4\mu$ and *btl-solver* can be attributed to the fact that our translation produces automata that are much smaller than the worst case but sometimes there is a very small counter-example which makes the btl-solver terminate quickly (for instance in $e5 \subseteq e6$).

### Satisfiability and Containment Modulo Schema.

The problem of satisfiability modulo schema is known to be EXPTIME-complete for the vast majority of queries found in practice, even for "simple" schema like DTD (Benedikt, Fan, and Geerts 2005). Specifically, the complexity depends

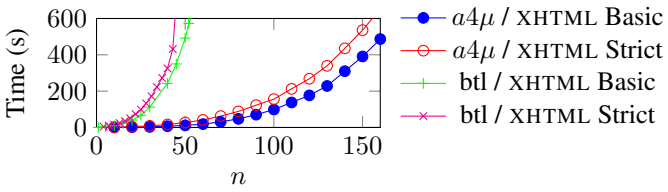



Figure 8: Satisfiability of $(//tr/*)^n$ with a XHTML schema.

both on the query size $n$ and on the schema size $m$. As noticed in (Libkin and Sirangelo 2010), a direct logical approach results in a $2^{\mathcal{O}(n \cdot m)}$ time complexity. This is interesting because once an automaton is built from the query, it can then be simply intersected with the automaton representing the constraint, which yields a better $\mathcal{O}((m+n) \cdot 2^n)$ time complexity (emptyness tests are linear for NDTA). For this reason, state-of-the-art implementations of the direct logical technique can hardly deal with recursive queries that require unfolding large schemas (like XHTML).

Figure 8 reports on satisfiability-testing times for the XPath $(//tr/*)^n$ under the XHTML Strict and Basic DTDs and shows that our method extends the envelope of practically solvable problem instances. In particular, the btl solver handles cases up to $n = 50$ in reasonable time while our method can handle up to $n = 150$. Figure 7 shows that we also speed up some of the feasible cases considerably (even with no schema involved).

## Conclusion

In this paper, we present a translation from an expressive tree logic to tree automata. We first introduce a notion of tree annotations and two properties on them: local consistency and consistency. We prove that local consistency is equivalent to the consistency. From there, we prove that the automaton construction is correct, and focus on a more parsimonious translation compared to the state-of-the-art.

The complexity of the construction is simply exponential in terms of the formula size. This is an improvement over previous translations, either in terms of the supported logical language expressivity or in terms of computational complexity of the construction. We explain how this construction can be implemented efficiently and provide a prototype implementation. To the best of our knowledge, this is the first implementation of a translation for such an expressive $\mu$-calculus. We have also carried out practical experiments for the static analysis of XPath queries under real-world schemas such as XHTML. Our prototype successfully solves practical instances that were beyond reach.

## References


Arroyuelo, D.; Claude, F.; Maneth, S.; Mkinen, V.; Navarro, G.; Nguyen, K.; Sirn, J.; and Vlimki, N. 2015. Fast in-memory XPath search using compressed indexes. *Software: Practice and Experience* 45(3).

Benedikt, M., and Cheney, J. 2010. Destabilizers and independence of XML updates. *PVLDB* 3(1):906–917.

Benedikt, M.; Fan, W.; and Geerts, F. 2005. XPath satisfiability in the presence of DTDs. In *PODS '05*. ACM Press.

Benzaken, V.; Castagna, G.; Nguyen, K.; and Siméon, J. 2013. Static and dynamic semantics of NoSQL languages. In *POPL'13*.

Björklund, H.; Gelade, W.; and Martens, W. 2010. Incremental xpath evaluation. *ACM Trans. Database Syst.* 35(4):29:1–29:43.

Bosch, M.; Genevès, P.; and Layaïda, N. 2015. Reasoning with style. IJCAI'15.

Calvanese, D.; Giacomo, G. D.; Lenzerini, M.; and Vardi, M. Y. 2010. Node selection query languages for trees. In *AAAI*.

Chekol, M. W.; Euzenat, J.; Genevès, P.; and Layaïda, N. 2012. SPARQL query containment under SHI axioms. In *AAAI*.

Doner, J. 1970. Tree acceptors and some of their applications. *Journal of Computer and System Sciences* 4.

Fard, A. M.; Mesbah, A.; and Wohlstadter, E. 2015. Generating fixtures for JavaScript unit testing. In *ASE'15*, 11 pages. IEEE Computer Society.

Franceschet, M. 2005. XPathMark: An XPath benchmark for the XMark generated data. In *Database and XML Technologies*, volume 3671 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Francis, N.; David, C.; and Libkin, L. 2011. A direct translation from XPath to nondeterministic automata. In *AMW'11*.

Friedmann, O., and Lange, M. 2010. A solver for modal fixpoint logics. *Electron. Notes Theor. Comput. Sci.* 262.

Genevès, P., and Gesbert, N. 2015. Xquery and static typing: tackling the problem of backward axes. In *ICFP'15*, 88–100.

Genevès, P.; Layaïda, N.; Schmitt, A.; and Gesbert, N. 2015. Efficiently Deciding $\mu$-calculus with Converse over Finite Trees. *ACM Trans. Comput. Log.*

Genevès, P.; Layaïda, N.; and Schmitt, A. 2007. Efficient static analysis of XML paths and types. In *PLDI '07*. New York, NY, USA: ACM Press.

Klarlund, N., and Møller, A. 2001. *MONA Version 1.4 User Manual*. BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus.

Klarlund, N.; Møller, A.; and Schwartzbach, M. I. 2001. MONA implementation secrets. In *CIAA '00*, volume 2088 of *LNCS*. London, UK: Springer-Verlag.

Kozen, D. 1983. Results on the propositional $\mu$-calculus. *Theoretical Computer Science* 27.

Libkin, L., and Sirangelo, C. 2008. Reasoning about xml with temporal logics and automata. In *LPAR '08*. Berlin, Heidelberg: Springer-Verlag.

Libkin, L., and Sirangelo, C. 2010. Reasoning about XML with temporal logics and automata. *J. Applied Logic* 8(2).

Marx, M. 2004. Conditional XPath, the first order complete XPath dialect. In *Symposium on Principles of Database Systems*, 13–22.

Pan, G.; Sattler, U.; and Vardi, M. Y. 2006. BDD-based decision procedures for the modal logic K. *Journal of Applied Non-classical Logics* 16(1-2).

Reichenbach, C.; Burke, M. G.; Peshansky, I.; and Raghavachari, M. 2009. Analysis of imperative XML programs. *Inf. Syst.* 34(7):624–642.

Schwentick, T. 2007. Automata for XML – a survey. *Journal of Computer and System Sciences* 73(3). Special Issue: Database Theory 2004.

Tanabe, Y.; Takahashi, K.; Yamamoto, M.; Tozawa, A.; and Hagiya, M. 2005. A decision procedure for the alternation-free two-way modal $\mu$-calculus. In *In TABLEAUX 2005*, volume 3702 of *LNCS*. London, UK: Springer-Verlag.

ten Cate, B.; Litak, T.; and Marx, M. 2010. Complete axiomatizations for XPath fragments. *J. Applied Logic* 8(2).

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

Ünel, G., and Toman, D. 2007. An incremental technique for automata-based decision procedures. In *CADE'07*.

Xu, Z. 2013. *Parametric Polymorphism for XML Processing Languages*. Theses, Université Paris-Diderot - Paris VII.

## Translation from NDTA to $\mathcal{L}_\mu$

NDTA can be encoded into $\mathcal{L}_\mu$ very easily. The resulting formula has only one n-ary fixpoint and only forward modalities (thus the cycle-freeness). This translation also proves the MSO-completeness of $\mathcal{L}_\mu$.

Let $(\mathcal{A}, \mathcal{Q}, \delta, \mathcal{E}, \#)$ be a NDTA. We associate to each state $s \in \mathcal{Q}$ to a variable $V_s$ and a formula $F_s$ and for each $l \in \mathcal{A}$ we associate a proposition $P_l$.

We use the function $t(s,i)$ to associate a state $s$ and a side $i \in \{1,2\}$ with a formula computing the states compatible with being the $i$-parent of $s$: $t(s,i) = \langle i \rangle V_s$ when $s \neq \#$ and $t(\#,i) = \neg \langle i \rangle \top$.

We also use a function $lbl(l)$ that associate a label $l$ with a formula testing this label (labels are mutually exclusive): $lbl(l) = P_l \bigwedge_{\substack{l' \in \mathcal{A} \\ l \neq l'}} \neg P_{l'}$.

In the automata, each state $s$ can be obtained through one of the transitions $(s_1, s_2, l, s) \in \delta$, therefore $F_s$ must impose that $s$ is compatible with being the $i$-parent of $s_1$ and with $l$. Thus, the formula $F_s$ associated with a state $s$ is

$$F_s = \bigvee_{(s_1,s_2,l,s)\in\delta} t(s_1,1) \wedge t(s_2,2) \wedge lbl(l)$$

Finally, the translation of the automaton into $\mathcal{L}_\mu$ is

$$\mu(V_s = F_s)\ in\ \neg\langle\bar{1}\rangle\top \wedge \neg\langle\bar{2}\rangle\top \wedge \bigvee_{s\in\mathcal{E}} V_s$$

## Proof of the theorem.

It is easy to see that consistency implies local consistency we now prove that local consistency implies consistency.

**The verification function $\mathcal{V}_\gamma$.** We consider a locally consistent annotation $\gamma$ and we will prove that $\gamma$ is a consistent annotation. The local consistency checks that an annotation is correct at range 0 *(Is the annotation correct with the atomic propositions and is it consistent with the existence and the type of neighbours?)*. We introduce the function $\mathcal{V}_\gamma$ checking the consistency of $\gamma$ at range $k$.

**Definition 10.** *We define $\mathcal{V}_\gamma$ a function taking a $\theta$-formula $\varphi$, a focused tree $\mathcal{T}$ ($\varphi$ is tested against $\mathcal{T}$), an integer $k$ (at which "range" do we check the formula) and returns a boolean. $\mathcal{V}_\gamma$ is well-defined because $k$ decrease at each step.*

- $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 0) = (\varphi \in \gamma(\mathcal{T}))$
- $\mathcal{V}_\gamma(\top, \mathcal{T}, k+1) = \top$
- $\mathcal{V}_\gamma(P, \mathcal{T}, k+1) = (P \in \gamma(\mathcal{T}))$
- $\mathcal{V}_\gamma(\neg\xi, \mathcal{T}, k+1) = \neg\mathcal{V}_\gamma(\xi, \mathcal{T}, k)$
- $\mathcal{V}_\gamma(\varphi_1 \wedge \varphi_2, \mathcal{T}, k+1) = \mathcal{V}_\gamma(\varphi_1, \mathcal{T}, k) \wedge \mathcal{V}_\gamma(\varphi_2, \mathcal{T}, k)$
- $\mathcal{V}_\gamma(\varphi_1 \vee \varphi_2, \mathcal{T}, k+1) = \mathcal{V}_\gamma(\varphi_1, \mathcal{T}, k) \vee \mathcal{V}_\gamma(\varphi_2, \mathcal{T}, k)$
- $\mathcal{V}_\gamma(\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi, \mathcal{T}, k+1) = \mathcal{V}_\gamma(exp(\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi), k)$
- $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, k+1) = \begin{cases} \mathcal{V}_\gamma(\varphi, \mathcal{T}\langle a \rangle, k) & when\ \mathcal{T}\langle a \rangle\ exists \\ \bot & when\ \mathcal{T}\langle a \rangle\ does\ not\ exist \end{cases}$

**Lemma 2.** *The function $k \to \mathcal{V}_\gamma(\varphi, \mathcal{T}, k)$ is constant.*

*Proof.* Let $\varphi \in cl(\theta)$ and a focused tree $\mathcal{T}$, we only need to prove that $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 1) = \mathcal{V}_\gamma(\varphi, \mathcal{T}, 0)$ by induction on the size of $\varphi$.

- $\mathcal{V}_\gamma(\top, \mathcal{T}, 1) = \top = \mathcal{V}_\gamma(\top, \mathcal{T}, 0)$ ;
- $\mathcal{V}_\gamma(P, \mathcal{T}, 1) = (P \in \gamma(\mathcal{T})) = \mathcal{V}_\gamma(P, \mathcal{T}, 0)$ ;
- $\mathcal{V}_\gamma(\varphi_1 \wedge \varphi_2), \mathcal{T}, 1) = \mathcal{V}_\gamma(\varphi_1, \mathcal{T}, 0) \wedge \mathcal{V}_\gamma(\varphi_2, \mathcal{T}, 0) = \mathcal{V}_\gamma(\varphi_1 \wedge \varphi_2, \mathcal{T}, 0)$
- $\mathcal{V}_\gamma(\varphi_1 \vee \varphi_2), \mathcal{T}, 1) = \mathcal{V}_\gamma(\varphi_1, \mathcal{T}, 0) \vee \mathcal{V}_\gamma(\varphi_2, \mathcal{T}, 0) = \mathcal{V}_\gamma(\varphi_1 \vee \varphi_2, \mathcal{T}, 0)$
- $\mathcal{V}_\gamma(\neg\xi), \mathcal{T}, 1) = \neg\mathcal{V}_\gamma(\xi, \mathcal{T}, 0) = \mathcal{V}_\gamma(\neg\xi, \mathcal{T}, 0)$
- $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 1) = \begin{cases} \mathcal{V}_\gamma(\varphi, \mathcal{T}\langle a \rangle, 0) & \text{when } \mathcal{T}\langle a \rangle \text{ exists} \\ \bot & \text{otherwise} \end{cases}$
  - If $\mathcal{T}\langle a \rangle$ does not exist then, since the annotation $\gamma$ is locally consistent, $\langle a \rangle \varphi \notin \gamma(\mathcal{T})$ and $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 1) = \bot = (\langle a \rangle \varphi \in \gamma(\mathcal{T})) = \mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 0)$.
  - It $\mathcal{T}\langle a \rangle$ does exist, then by local consistency of $\gamma$ we have $\Delta_a(\gamma(\mathcal{T}), \gamma(\mathcal{T}\langle a \rangle))$ which implies $(\varphi \in \gamma(\mathcal{T}\langle a \rangle)) = (\langle a \rangle \varphi \in \gamma(\mathcal{T}))$ and thus: $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 1) = \mathcal{V}_\gamma(\varphi, \mathcal{T}\langle a \rangle, 0) = (\varphi \in \gamma(\mathcal{T}\langle a \rangle)) = \langle a \rangle \varphi \in \gamma(\mathcal{T}) = \mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 0)$
- $\mathcal{V}_\gamma(\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi, \mathcal{T}, 1) = \mathcal{V}_\gamma(\exp(\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi), 0) = (\exp(\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi) \in \gamma\mathcal{T}) = (\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi \in \gamma\mathcal{T}) = \mathcal{V}_\gamma(\mu(X_i = \varphi_i)_{i\in I}\ in\ \psi, \mathcal{T}, 0)$

□

**Equivalence between $\mathcal{V}_\gamma$ and $[\![\varphi]\!]$.**

**Lemma 3.** *For every focused tree $\mathcal{T}$ and every $\theta$-formula $\varphi$, when all paths of $\varphi$ longer than $k$ are not valid paths of $\mathcal{T}$ then $\mathcal{V}_\gamma(\varphi, \mathcal{T}, k) \Leftrightarrow \mathcal{T} \in [\![\varphi]\!]$.*

*Proof.* We show that recursively on $k$. The empty path is always valid therefore $k > 0$. The constraint on paths holds since recursive calls on $\mathcal{V}_\gamma$ corresponds to path of size 1. Therefore, if there is a recursive call on $\mathcal{V}_\gamma(\psi, \mathcal{T}', k-1)$ all valid paths of $\psi$ on $\mathcal{T}'$ with size $s$ correspond to valid paths of $\varphi$ on $\mathcal{T}$ with size $s+1$.

- $\mathcal{V}_\gamma(P, \mathcal{T}, k+1) = (P \in \gamma(\mathcal{T}) = (\mathcal{T} \in [\![\varphi]\!])$;
- $\mathcal{V}_\gamma(\varphi_1, \wedge, \varphi_2), \mathcal{T}, k+1) = \mathcal{V}_\gamma(\varphi_1, \mathcal{T}, 2) \wedge \mathcal{V}_\gamma(\varphi_2, \mathcal{T}, k) = (\mathcal{T} \in [\![\varphi_1]\!]) \wedge (\mathcal{T} \in [\![\varphi_2]\!]) = (\mathcal{T} \in [\![\varphi_1]\!] \cap [\![\varphi_2]\!]) = (\mathcal{T} \in [\![\varphi_1 \wedge \varphi_2]\!])$;
- we use the same argument for $\varphi_1 \vee \varphi_2$, $\neg\xi$, $\top$
- if $\mathcal{T}\langle a \rangle$ is defined then $\langle a \rangle$ is a valid path of $\mathcal{T}$ and thus $k \geq 1$. We have
  $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, k) = \mathcal{V}_\gamma(\varphi, \mathcal{T}\langle a \rangle, k-1)$ and
  $\mathcal{V}_\gamma(\varphi, \mathcal{T}\langle a \rangle, k-1) = (\mathcal{T}\langle a \rangle \in [\![\varphi]\!]) = (\mathcal{T}\langle a \rangle \in [\![\varphi]\!]) = (\mathcal{T}\langle a \rangle \langle \bar{a} \rangle \in [\![\langle a \rangle \varphi]\!]) = (\mathcal{T} \in [\![\langle a \rangle \varphi]\!])$.
- if $\mathcal{T}\langle a \rangle$ is not defined then $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, k) = \bot$ and $\mathcal{T} \notin [\![\langle a \rangle \varphi]\!]$.

**Conclusion of the proof.** There is no valid path $p$ of $\varphi$ such that $p$ is of size greater than $c(\varphi, \mathcal{T})$ and thus $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 0) = \mathcal{V}_\gamma(\varphi, \mathcal{T}, c(\varphi, \mathcal{T}) + 1) = (\mathcal{T} \in [\![\varphi]\!])$. Since $\mathcal{V}_\gamma(\varphi, \mathcal{T}, k) = \mathcal{V}_\gamma(\varphi, \mathcal{T}, 0)$ we have $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 0) = \mathcal{T} \in [\![\varphi]\!]$

For all $\varphi \in cl(\theta)$, $(\varphi \in \gamma(\mathcal{T})) = \mathcal{V}_\gamma(\varphi, \mathcal{T}, 0) = (\mathcal{T} \in [\![\varphi]\!])$, which means $\gamma(\mathcal{T})$ is a consistent type and thus $\gamma$ is a consistent annotation. □