An Efficient Translation from a Two-Way μ-Calculus of Finite Trees into Tree Automata

Louis Jachiet, Pierre Genevès, Nabil Layaïda

An Efficient Translation from a Two-Way μ-Calculus of Finite Trees into Tree Automata

Louis Jachiet[123], Pierre Genevès[231], and Nabil Layaïda[321]
firstname.lastname@inria.fr

[1] Univ. Grenoble Alpes
[2] CNRS, LIG
[3] Inria

Abstract. We present a direct translation from a sub-logic of μ-calculus to non-deterministic binary automata of finite trees. The logic is an alternation-free modal μ-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). The size of the generated automaton (the number of transitions) is bounded by 2^n where n is the size of a Fischer-Ladner closure of the formula. This is an improvement over previous translations in 2^{n^2}. We have implemented our translation. In practice, our prototype effectively decides static analysis problems that were beyond reach, such as the XPath containment problem with DTDs of significant size.

1 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 MSO-complete (Monadic Second Order) logical language. 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, as pointed out in e.g. [17] and [20]. Implementations of satisfiability-testing algorithms for expressive logics rarely rely on automata-based techniques. Notable exceptions include MONA [12,13] for the weak monadic second-order logic of two successors (WS2S) [6,19] and MLSolver [9] for the full μ-calculus (without converse modalities) [14]. In general, however, automata-based decision procedures implementations are often outperformed by alternative techniques such as tableau methods, as found in e.g. [11,17,18]. Such techniques try to avoid one of the main weakness 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, as pointed out in [16]. Another application is the static typing of tree manipulating functional programs. 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 [3] need to produce a tree automaton from some logical language representing statements à la XPath [5]. Finally, queries evaluators seeking performance can benefit from such a translation by incorporating the automata construction directly in their compilers [1]. 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 fragment of the μ-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 [4,10,16]. An efficient decision procedure for testing the satisfiability of this logic has been successfully designed and implemented in [11]. 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 automaton transitions.

Related Work. Several works addressed the translation of formulae into tree automata. The seminal work on MONA [12,13] developed implementation techniques for WS2S which is succinct, but whose satisfiability is non-elementary. The more closely related work [15] introduced a translation from CXPath to tree automata in $2^{O(n)}$. This work was first introduced in [15] and further developed in [16] and [8]. However, CXPath represents a strict subset of μ-calculus and weaker than the tree logic used here. In addition, their translation produce unranked tree automata for which efficient implementations are notoriously lacking (none have been reported). Closer to this work, [4] presented a translation from μ-calculus to tree automata. However their translation has higher complexity than ours (2^{n^2} states) and has no reported implementation.

Contributions. Our contributions are twofold. First, we tighten the complexity bound obtained by [4] from $O(2^{n^2})$ automata states to $O(2^n)$ where n is proportional to the formula size. We also show that the number of transitions is smaller than $O(2^n)$. Second, 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.

Paper Outline. We introduce in Section 2 the tree logic: its tree models, formulae and their interpretations, fixpoints and some other necessary technical material for the translation. Section 3 introduces the notion of tree annotations in order to capture translation consistency both local and global. Then, we proceed in Section 4 by describing a globally consistent translation technique from formulae of a tree-logic with backward modalities to a tree automaton. In particular, we explain how states and transitions are extracted and how the overall automaton size is obtained. In Section 5, we report on experimental results for the static analysis of XPath queries under real-world schemas (such as XHTML) before concluding in Section 6.

2 Logical context

2.1 Focused trees

The trees we consider are binary trees (our results transfer to unranked trees thanks to the well-known "first-child next-sibling" bijective mapping). Focused trees are trees with the additional information of which node we are focused on. We have four "programs" to change the focus in a focused tree: $\langle 1\rangle$,$\langle 2\rangle$,$\langle \bar{1}\rangle$,$\langle \bar{2}\rangle$. If $\mathcal{T}$ is 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.

2.2 Formulae

The logic we consider is a fragment of the alternation free modal μ-calculus with a few additional restrictions (presented in section 2.6) that make the least and greatest fixpoints coincide. The grammar of our formulae is given below. The boolean functions we consider are arbitrary functions of the form $\{0,1\}^k \to \{0,1\}$. Typical such functions are $f(\varphi_1, \varphi_2) = \varphi_1 \vee \varphi_2$, $f(\varphi_1, \varphi_2) = \varphi_1 \wedge \varphi_2$, $f(\varphi) = \neg\varphi$ and $f() = \top$.

$\varphi ::=$	formula	$\mid P_i$	atomic proposition
$\mid f(\varphi_1, \ldots, \varphi_k)$	f is a boolean function	$\mid X$	variable
$\mid \mu\overline{X_i = \varphi_i}\ in\ \psi$	least n-ary fixpoint	$\mid \langle a\rangle\,\varphi$	modality

2.3 Interpretation of formulae

We consider a boolean function f of arity k and sets $S_1, \ldots, S_n$ of focused trees. We define $\chi(x, E) = 1$ when $x \in E$ and $\chi(x, E) = 0$ otherwise. We then define $f(S_1, \ldots, S_k)$ as $\{x \in \mathcal{F} \mid f(\chi(x, S_1), \ldots, \chi(x, S_n))\}$. This extends naturally the usual interpretation ($[\![\varphi_1 \wedge \varphi_2]\!]V = [\![\varphi_1]\!]V \cap [\![\varphi_2]\!]V$, $[\![\varphi_1 \vee \varphi_2]\!]V = [\![\varphi_1]\!]V \cup [\![\varphi_2]\!]V$ and $[\![\neg\varphi]\!] = \mathcal{F} \setminus [\![\varphi]\!]V$) to general boolean formulae. The interpretation of formulae is the following:

$$
\begin{aligned}
[\![X]\!]V &= V(X) \\
[\![P_i]\!]V &= \{\, \mathcal{T} \in \mathcal{F} \mid P_i \text{ is true on the node focused by } \mathcal{T}\} \\
[\![f(\varphi_1, \ldots, \varphi_n)]\!]V &= f\big([\![\varphi_1]\!]V, \ldots, [\![\varphi_n]\!]V\big) \\
[\![\langle a\rangle\, \varphi]\!]V &= \{\mathcal{T}\,\langle a\rangle \mid \mathcal{T} \in [\![\varphi]\!]V \wedge \mathcal{T}\,\langle a\rangle \text{ is defined}\} \\
[\![\mu\overline{X_i = \varphi_i}\ in\ \psi]\!]V &= [\![\psi]\!]V[X_i \to U_i]
\end{aligned}
$$

Where $S = \{(T_i) \in \mathcal{P}(\mathcal{F})^n \mid [\![\varphi_j]\!]V[X_i \to T_i] \subset T_j\}$ and $U_j = \bigcap_{(T_i)\in S} T_j$.

2.4 Unfolding fixpoints

Definition 1 *A subformula ψ of a formula φ is* guarded *in φ if there is a subformula $\langle a\rangle\, \kappa$ of φ such that ψ is a subformula of κ.*

Restriction 1 *We impose on formulae that for every fixpoint $\mu\overline{X_i = \varphi_i}$ in ψ, the variables X_i are guarded in ψ.*

Definition 2 *The* unfolding *of a formula φ is an equivalent formula φ' where unguarded fixpoints are expanded (each occurrence of a variable in a fixpoint has to be guarded, so fixpoints are expanded, at most, once). In the resulting formula $unf(\varphi)$ all fixpoints are guarded.*

$$
\begin{aligned}
unf(P_i)) &= P_i & unf(f(\varphi_1, \ldots, \varphi_n)) &= f(unf(\varphi_1), \ldots, unf(\varphi_n)) \\
unf(\langle a\rangle\, \varphi) &= \langle a\rangle\, \varphi & unf(\mu\overline{X_i = \varphi_i}\ in\ \psi) &= unf(\psi_{[X_i/\mu\overline{X_i=\varphi_i}\ in\ \psi]})
\end{aligned}
$$

2.5 Path

Definition 3 *A path $p = \langle a_1\rangle\, \langle a_2\rangle \ldots \langle a_n\rangle$ is* valid *on a focused tree $\mathcal{T}$ if for $1 \leq i \leq n$, $(\ldots(\mathcal{T}\,\langle a_1\rangle)\ldots)\,\langle a_i\rangle$ is defined. The set of paths for a formula is:*

$$
\mathcal{P}(P_i) = \emptyset \qquad\qquad \mathcal{P}(f(\varphi_1, \ldots, \varphi_n)) = \textstyle\bigcup_{i=1}^{n} \mathcal{P}(\varphi_i)
$$

$$
\mathcal{P}(\langle a\rangle\, \varphi) = \{\langle a\rangle\} \cup \{\langle a\rangle\, p \mid p \in \mathcal{P}(unf(\varphi))\}
$$

2.6 Cycle-free formulae

Definition 4 *Because our underlying models are binary trees, a path can only give the focus twice on the same node if there is a pattern* $\langle a\rangle\,\langle \bar{a}\rangle$ *or* $\langle \bar{a}\rangle\,\langle a\rangle$*. Such a pattern is called a cycle. A formula is* cycle-free *if there is a finite bound on the number of cycles the paths of this formula can contain.*

Definition 5 *Let* φ *be a cycle-free formula and* $\mathcal{T}$ *be a fixed focused binary tree, we take the smallest* n *such that no path of* φ *has* n *cycles and we define* $n(\varphi,\mathcal{T}) = n \times |\mathcal{T}|$*.*

Lemma 1 *Any path of* φ *that is longer than* $n(\varphi,\mathcal{T})$ *is not a valid path in* $\mathcal{T}$*.*

Proof. A valid path with no cycle is smaller than the number of nodes. Otherwise, it would give the focus twice on the same node and there would be a cycle. Let p be a valid path of φ. We can split p into n paths with no cycle pattern. Each of these paths is valid somewhere in the tree and does not contain any cycle, therefore each of these paths is smaller than $|\mathcal{T}|$: p is either smaller than $n \times \mathcal{T}$ or invalid. □

Restriction 2 *We impose that all formulae are cycle-free.*

2.7 $\mathcal{L}$ean, type, consistent type, Δ_a-compatibility

Definition 6 *The* $\mathcal{L}ean$ *is defined for an unfolded formula. It is a small variation of the classical Fischer-Ladner closure. The* $\mathcal{L}ean$ *can be defined as*[4]*:*

$$\mathcal{L}ean\,(P_i) = \{P_i\} \qquad \mathcal{L}ean\,(f(\varphi_1,\ldots,\varphi_k)) = \cup_{i=1}^{k}\mathcal{L}ean\,(\varphi_i)$$

$$\mathcal{L}ean\,(\langle a\rangle\,\varphi) = \{\langle a\rangle\,\varphi\} \cup \mathcal{L}ean\,(unf(\varphi))$$

Lemma 2 *For any unfolded formula* ψ*,* $\mathcal{L}ean(\psi)$ *is finite.*

Proof. The set $\mathcal{L}\text{ean}(\psi)$ contains the atomic propositions appearing in ψ and the formulae of the form $\langle a\rangle\,\varphi$ where $\langle a\rangle\,\varphi$ is a subformula of ψ or a subformula of an unfolded subformula of ψ. □

Remark 1 *For the rest of the paper, we fix the formula* ξ *and we write* $\mathcal{L}ean$ *for* $\mathcal{L}ean(unf(\xi))$*,* n *for the size of* $\mathcal{L}ean$ *and* $\omega_1,\ldots,\omega_n$ *for* $\mathcal{L}ean$*. A formula* φ *for which* $\mathcal{L}ean(unf(\varphi))$ *is included in* $\mathcal{L}ean$ *is called a* $\mathcal{L}ean$ *formula.*

Lemma 3 *For a* $\mathcal{L}ean$*-formula* φ *there is a* f *such that* $[\![\varphi]\!] = [\![f(\omega_1,\ldots,\omega_n)]\!]$*.*

Proof. For a formula κ with no free variables, $\text{unf}(\kappa)$ is necessarily of one the subforms: P_j, $\langle a\rangle\,\psi$ or $f(\psi_1,\ldots,\psi_n)$ where the ψ_i are also of one of these three forms.

[4] We recall that an unfolded formula has no unguarded fixpoint.

For $\varphi = f_p(\varphi_1, \ldots, f_i(\varphi_{i,1}, \ldots, \varphi_{i,k}), \ldots, \varphi_l)$, we can combine f_p and f_i into an equivalent f_c ($f_c(v_1, \ldots, v_{i,1}, \ldots, v_{i,k}, \ldots, v_l) = f_p(v_1, \ldots, f_i(vi, 1, \ldots, v_{i,k}), \ldots, v_l)$). and for φ not a function, we can use the identity function to get the semantically equivalent $(x \to x)(\varphi)$.

In any case $\mathrm{unf}(\kappa)$ can always be transformed in an equivalent $f(\varphi_1, \ldots, \varphi_l)$ where f is a boolean function and the φ_j are unguarded subformulae of φ of the form P_j and $\langle a \rangle \psi$. φ is a $\mathcal{L}$ean formula, so each φ_j is necessarily a ω_k for some k. For φ a function using a subset of $\{\omega_1, \ldots, \omega_n\}$ as arguments we can add arguments to f and permute them to get a f such that $\mathrm{unf}(\kappa) = f(\omega_1, \ldots, \omega_n)$. □

Remark 2 *We introduce the variables $(V_1, \ldots, V_n)$ with $[\![V_i]\!] = [\![\omega_i]\!]$. For $i \in \{1..n\}$, if $\omega_i = P_j$ then $V_i = P_j$ otherwise, let a_i and f_i be such that $[\![\omega_i]\!] = [\![\langle a_i \rangle f_i(V_1, \ldots, V_n)]\!]$ and we state $V_i = \langle a \rangle f_i(V_1, \ldots, V_n)$. For the formula ξ let f_ξ be such that $[\![\xi]\!] = [\![f_\xi(V_1, \ldots, V_n)]\!]$.*

Definition 7 *A* type *is an element of $\{0,1\}^n$. For a type $\mathbf{t} = (\mathbf{t}_1, \ldots, \mathbf{t}_n)$ and $i \in 1..n$, $\delta_i(\mathbf{t})$ is true when $\mathbf{t}_i = 1$. For a type $\mathbf{t}$ and a boolean function f of arity n, we write $f(\mathbf{t})$ for $f(\delta_1(\mathbf{t}), \ldots, \delta_n(\mathbf{t}))$. A type $\mathbf{t}$ is said consistent with a focused tree $\mathcal{T}$ when $\forall i \in 1..n$ $(\delta_i(\mathbf{t}) \Leftrightarrow \mathcal{T} \in [\![V_i]\!])$.*

We denote by $\mathcal{L}$ the set of i such that V_i represents an atomic proposition; F_a the set of V_i using the modality a, $F_a = \{i \mid (V_i = \langle a \rangle f_i(V)) \in \mathcal{L}\mathrm{ean}\}$.

Definition 8 *Let $\widetilde{1} = \bar{2}$ and $\widetilde{2} = \bar{1}$, let $\mathbf{x}$ and $\mathbf{y}$ be two types, $a \in \{1,2\}$, the compatibility operator $\Delta_a(\mathbf{x}, \mathbf{y})$ is: $\Delta_a(\mathbf{x}, \mathbf{y}) = (\forall i \in F_a\ \delta_i(\mathbf{x}) = f_i(\mathbf{y})) \wedge (\forall i \in F_{\bar{a}}\ \delta_i(\mathbf{y}) = f_i(\mathbf{x})) \wedge (\forall i \in F_{\widetilde{a}}\ \delta_i(\mathbf{y}) = 0)$.*

Remark 3 *Δ_a is such that if we have $\Delta_a(t_p, t_c) \wedge \Delta_a(t'_p, t_c) \wedge \Delta_a(t_p, t'_c)$ then we necessarily have $\Delta(t'_p, t'_c)$.*

3 Annotation of trees

We introduce the notions of locally and globally consistent tree annotations and prove that they are equivalent. Local consistency will be used in Section 4 to derive automaton construction, while global consistency will help in establishing that the automaton captures the set of trees that are models of the formula.

Definition 9 *An* annotation *of a finite tree $\mathcal{T}$ is a function from the nodes of $\mathcal{T}$ to types. An annotation γ is consistent when each node $\mathcal{N}$ is associated with a consistent type $\gamma(\mathcal{N})$. Note that a given tree has only one consistent annotation.*

Definition 10 *We say that an annotation γ of the tree $\mathcal{T}$ is locally consistent when:*

- *for each node $\mathcal{N}$ and each atomic proposition P_i, $(\mathcal{N} \in [\![P_i]\!]) \Leftrightarrow \delta_i(\gamma(\mathcal{N}))$;*
- *if $\mathcal{N}_c$ is the a-child of $\mathcal{N}_p$, then we have $\Delta_a(\gamma(N_p), \gamma(N_c))$;*
- *if $\mathcal{N}$ has no a-child then there are no $i \in F_a$ such that $\delta_i(\gamma(\mathcal{N}))$;*
- *if $\mathcal{N}$ has no parent then there are no $i \in F_{\bar{1}} \cup F_{\bar{2}}$ such that $\delta_i(\gamma(\mathcal{N}))$.*

3.1 The verification function $\mathcal{V}_\gamma$

We consider γ, a locally consistent annotation. We want to prove that a locally consistent annotation is a consistent annotation. The local consistency checks that an annotation is correct at range 1 *(Is the annotation correct with the atomic propositions and is it consistent with the existence and the type of neighbours?)*. We build a function $\mathcal{V}_\gamma$ checking the consistency at range k (*Is the annotation valid if we cross, at most, k modalities?*).

Definition 11 *We define $\mathcal{V}_\gamma$ a function taking a $\mathcal{L}$ean formula φ, a tree $\mathcal{T}$ focused on a node t (φ is tested against $\mathcal{T}$), an integer k (at which "range" do we check the formula) and returning a boolean. We define $\mathcal{V}_\gamma$ by induction on k decreasing and on the size of the formula φ.*

- *If $\varphi = P_j$ there is a unique i such that $[\![V_i]\!] = [\![P_j]\!]$, $\mathcal{V}_\gamma(\varphi, \mathcal{T}, k) = (\delta_i(\gamma(t)))$*
- $\mathcal{V}_\gamma(f(\varphi_1, \ldots, \varphi_n), \mathcal{T}, k) = f(\mathcal{V}_\gamma(\varphi_1, \mathcal{T}, k), \ldots, \mathcal{V}_\gamma(\varphi_n, \mathcal{T}, k))$
- $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, k) = \begin{cases} \langle a \rangle \varphi \in \gamma(t) & \text{if } k = 0 \\ \mathcal{V}_\gamma(unf(\varphi), \mathcal{T}\langle a \rangle, k-1) & \text{if } k > 0 \wedge \mathcal{T}\langle a \rangle \text{ exists} \\ 0 & \text{if } k > 0 \wedge \mathcal{T}\langle a \rangle \text{ does not exist} \end{cases}$

Lemma 4 *The function $k \to \mathcal{V}_\gamma(\varphi, \mathcal{T}, k)$ is constant.*

Proof. Let φ be a $\mathcal{L}$ean formula and $\mathcal{T}$ a tree focused on the node 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 φ.

- $\mathcal{V}_\gamma(P_j, \mathcal{T}, 1) = \delta_i(\gamma(t)) = \mathcal{V}_\gamma(P_j, \mathcal{T}, 0)$ (for the i such that $[\![V_i]\!] = [\![P_j]\!]$) ;
- $\mathcal{V}_\gamma(f(\varphi_1, \ldots, \varphi_k), \mathcal{T}, 1) = f(\mathcal{V}_\gamma(\varphi_1, \mathcal{T}, 1), \ldots, \mathcal{V}_\gamma(\varphi_k, \mathcal{T}, 1)) =$
  $f(\mathcal{V}_\gamma(\varphi_1, \mathcal{T}, 0), \ldots, \mathcal{V}_\gamma(\varphi_k, \mathcal{T}, 0)) = \mathcal{V}_\gamma(f(\varphi_1, \ldots, \varphi_k), \mathcal{T}, 0)$;
- $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 1) = \begin{cases} \mathcal{V}_\gamma(\text{unf}(\varphi), \mathcal{T}\langle a \rangle, k-1) & \text{when } \mathcal{T}\langle a \rangle \text{ exists} \\ 0 & \text{otherwise} \end{cases}$

  Let $i \in F_a$ be such that $V_i = \langle a \rangle \varphi$, we have $\text{unf}(\varphi) = f_i(\varphi_1, \ldots, \varphi_n)$. The annotation γ is consistent, if $\mathcal{T}\langle a \rangle$ does not exist then $\delta_i(\gamma(\mathcal{T})) = 0$ and $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 1) = 0 = \delta_i(\gamma(\mathcal{T})) = \mathcal{V}_\gamma(\varphi, \mathcal{T}, 0)$.
  If $\mathcal{T}\langle a \rangle$ does exist, $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 1) = \mathcal{V}_\gamma(\text{unf}(\varphi), \mathcal{T}\langle a \rangle, 0) =$
  $f_i(\mathcal{V}_\gamma(\omega_1, \mathcal{T}\langle a \rangle, 0), \ldots, \mathcal{V}_\gamma(\omega_n, \mathcal{T}\langle a \rangle, 0))) = f_i(\delta_1(\gamma(T\langle a \rangle)), \ldots, \delta_n(\gamma(T\langle a \rangle))) =$
  $f_i(\gamma(\mathcal{T}\langle a \rangle)) = \delta_i(\gamma(\mathcal{T})) = \mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 0)$. The equality $f_i(\gamma(\mathcal{T}\langle a \rangle)) = \delta_i(\gamma(\mathcal{T}))$ stands by definition of $\Delta_a(\gamma(\mathcal{T}), \gamma(\mathcal{T}\langle a \rangle))$.

□

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

Lemma 5 *For every focused tree $\mathcal{T}$ and every $\mathcal{L}$ean-formula φ we have $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 0) \Leftrightarrow \mathcal{T} \in [\![\varphi]\!]$.*

Proof. We now show that for $\varphi, k, \mathcal{T}$ when all paths of $\mathcal{P}(\varphi)$ longer than k are not valid paths of $\mathcal{T}$ then $\mathcal{V}_\gamma(\varphi, \mathcal{T}, k) = \chi(\mathcal{T}, [\![\varphi]\!])$.

- $\mathcal{V}_\gamma(P_j, \mathcal{T}, k) = \delta_i(\gamma(t)) = (\mathcal{T} \in [\![\varphi]\!])$ (for i such that $[\![V_i]\!] = [\![P_j]\!]$);
- $\mathcal{V}_\gamma(f(\varphi_1, \ldots, \varphi_k), \mathcal{T}, k) = f(\mathcal{V}_\gamma(\varphi_1, \mathcal{T}, k), \ldots, \mathcal{V}_\gamma(\varphi_k, \mathcal{T}, k))$
  $= f(\chi(\varphi_1, \mathcal{T}, k), \ldots, \chi(\varphi_k, \mathcal{T}, k)) = \chi(\mathcal{T}, [\![f(\varphi_1, \ldots, \varphi_k)]\!])$;
- if $\mathcal{T}\langle a \rangle$ is defined then $\langle a \rangle$ is a valid path of $\mathcal{T}$ and $k \geq 1$. We have $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, k) = \mathcal{V}_\gamma(\mathrm{unf}(\varphi), \mathcal{T}\langle a \rangle, k-1)$ and $\mathcal{V}_\gamma(\mathrm{unf}(\varphi), \langle a \rangle \mathcal{T}, k-1) = \chi(\mathcal{T}\langle a \rangle, [\![\mathrm{unf}(\varphi)]\!]) = \chi(\mathcal{T}\langle a \rangle, [\![\varphi]\!]) = \chi(\mathcal{T}\langle a \rangle \langle \bar{a} \rangle, [\![\langle a \rangle \varphi]\!]) = \chi(\mathcal{T}, [\![\langle a \rangle \varphi]\!])$. The constraint on paths holds, a path of $\mathcal{P}(\mathrm{unf}(\varphi))$ on $\mathcal{T}\langle a \rangle$ of size k is a valid path of $\mathcal{P}(\varphi)$ on $\mathcal{T}\langle a \rangle$ of size k and therefore a valid path of size k for φ on $\mathcal{T}$;
- if $\mathcal{T}\langle a \rangle$ is not defined then $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, k) = 0$ and $\neg\chi(\mathcal{T}, [\![\langle a \rangle \varphi]\!])$.

There is no valid path $p \in \mathcal{P}(\varphi)$ such that p is of size greater than $n(\varphi, \mathcal{T})$. Thus $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 0) = \mathcal{V}_\gamma(\varphi, \mathcal{T}, n(\varphi, \mathcal{T}) + 1) \Leftrightarrow \mathcal{T} \in [\![\varphi]\!]$ so $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 0) \Leftrightarrow \mathcal{T} \in [\![\varphi]\!]$.

□

Theorem 1 (Local consistency is consistency) *Given a tree $\mathcal{T}$, an annotation is locally consistent iff it is globally consistent.*

Proof. $\Rightarrow$ Consistency of a type $\mathbf{t}$ on a focused tree $\mathcal{T}$ being defined as $\mathbf{t} \in [\![\mathcal{T}]\!]$ lemma 5 proves the global consistency of locally consistent annotations.

$\Leftarrow$ By construction of the local consistency, it is implied by the global consistency. □

4 Automaton construction

A first idea for an automaton that checks a formula is to have types as states and make the transitions enforce the local consistency. This leads to a bottom-up non-deterministic tree automaton that has 2^n states and no more than $(2^n)^3$ transitions. We present here an improved construction of a bottom-up non-deterministic tree automaton that is much smaller ($O(2^n)$ transitions in the worst case) and more prone to optimization.

4.1 States of the automaton

States of our automaton are sets of types with a "side" (1 or 2) plus a unique final state $\mathcal{F}$. The information contained in a state (S, i) (where S is a set of types and $i \in \{1, 2\}$) is that S represents a set of possible types for the i-parent of the current node.

Definition 12 *For $i \in \{1, 2\}$, the interface i of a type $\mathbf{y}$ is the set $\mathcal{S}_i(\mathbf{y})$ of types Δ_i-compatible with $\mathbf{y}$ and compatible with having a i-parent and $\#_i$ as the set of types having no i-child. $\mathcal{S}_i(\mathbf{y}) = \{x \mid \Delta_i(x, \mathbf{y})\}$ and $\#_i = \{x \mid \forall v \in F_i\ \delta_v(x) = 0\}$.*

Definition 13 $\mathcal{F} = \{t \mid f_\xi(t) \wedge (\forall i \in (F_{\bar{1}} \cup F_{\bar{2}}), \delta_i(t) = 0)\}$ *is the set of types that are compatible with being a root and a solution of ξ.*

Definition 14 *Let $C_i = \{\mathcal{S}_i(\mathbf{t}) \mid \mathbf{t}$ type$\} \cup \#_i$, the set of states of our automaton is $\mathcal{Q} = \{(E, i) \mid E \in C_i\} \cup \{(\mathcal{F}, 0)\}$.*

Remark 4 *As a consequence of remark 3, given $i \in \{1, 2\}$ and two types y_1, y_2, we have either $\mathcal{S}_i(y_1) \cap \mathcal{S}_i(y_2) = \emptyset$ or $\mathcal{S}_i(y_1) = \mathcal{S}_i(y_2)$.*

4.2 Transitions

We use the representation $e_1, e_2 \xrightarrow{l} e_3$ to indicate that there is a transition where e_1 is the state of the 1-child, e_2 is the state of the 2-child, l is the label and e_3 is the state of the parent. $(\#_i, i)$ represents the state of a leaf that is a i-child.

The alphabet of our automaton is the powerset of the set $\mathcal{L}$ of atomic propositions appearing in ξ. A node n is labelled with l when $\forall p \in \mathcal{L}\ (p \in l \Leftrightarrow n \in [\![p]\!])$

Definition 15 *A type $\mathbf{t}$ forces the value of each atomic proposition appearing in ξ. We write $\mathcal{L}(t)$ for the set of atomic proposition implied by $\mathbf{t}$.*

Let $e_1 = (E_1, 1)$, and $e_2 = (E_2, 2)$ for each $\mathbf{t} \in E_1 \cap E_2$ and for $i \in \{1, 2\}$, $e_1, e_2 \xrightarrow{\mathcal{L}(\mathbf{t})} \mathcal{S}_i(\mathbf{t})$ is a transition. We also have a transition $e_1, e_2 \xrightarrow{\mathcal{L}(\mathbf{t})} \mathcal{F}$ for each $\mathbf{t} \in \mathcal{F} \cap E_1 \cap E_2$. We say that those transitions are built with the type $\mathbf{t}$.

Lemma 6 *A tree is accepted if and only if it satisfies the formula.*

Proof. $\Rightarrow$ Let $\mathcal{T}$ be a tree and suppose that $\mathcal{T}$ is accepted. There is a valid run η, let $\eta(n)$ be the state the run associates with the node n. We will introduce an annotation τ and show that τ is locally consistent.

For all non root node n, the state $\eta(n)$ was obtained using a transition of the form $(e_1, 1), (e_2, 2) \xrightarrow{\mathcal{L}(t)} (\mathcal{S}_i(t), i)$ with $\mathbf{t} \in e_1 \cap e_2$. Let $\tau(n)$ be such a $\mathbf{t}$. The run is accepting, so the root node r is associated with $\eta(r) = \mathcal{F}$. $\mathcal{F}$ comes from a transition $(e_1, 1), (e_2, 2) \xrightarrow{\mathcal{L}(t)} \mathcal{F}$ and that ensures there is a $\mathbf{t}$ in $e_1 \cap e_2 \cap \mathcal{F}$, let $\tau(r)$ be such a $\mathbf{t}$.

τ is locally consistent: A node n is labelled with $\mathcal{L}(\tau(n))$. Let n_i be the i-child of n_p and let $(E_i, i) = \eta(n_i)$, we have $\tau(n) \in E_i$ and $E_i = \mathcal{S}_i(\tau(n_i))$. Thus $\tau(n) \in \mathcal{S}_i(\tau(n_i))$ which means $\Delta_i(\tau(n), \tau(n_i))$. Let n be a node, if n has no i-child, $\tau(n) \in \#_i$ which implies $\delta_i(\tau(n)) = 0$ for $i \in F_i$. Finally, the root r is associated with $\tau(r) \in \mathcal{F}$ and therefore $\delta_i(\tau(r)) = 0$ for $i \in F_{\bar{1}} \cup F_{\bar{2}}$.

$\Leftarrow$ Let $\mathcal{T}$ be a tree and suppose that $\mathcal{T}$ satisfies the formula ξ. We consider the annotation γ of $\mathcal{T}$ that associates each nodes n with its consistent type $\gamma(n)$. The function ρ that associates each node n of $\mathcal{T}$ with $\rho(n) = \mathcal{S}_i(\gamma(n))$ when n is a i-child and with $\rho(n) = \mathcal{F}$ when n is the root node is an accepting run. □

4.3 Size of the automaton

The number of sets $\mathcal{S}_i(t)$ depends only on which formulae of $F_i \cup F_{\bar{i}}$ are true. Therefore, the number of distinct $\mathcal{S}_a$ classes is bounded by $2^{|F_a \cup F_{\bar{a}}|}$. 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 three ($\#_1$, $\#_2$ and $\mathcal{F}$). We have $2^{|F_1 \cup F_{\bar{1}}|} + 2^{|F_2 \cup F_{\bar{2}}|} \leq 2^n$ (where $n = |\mathcal{L}\text{ean}|$) so the automaton has, at most, $3 + 2^n$ states.

Each transition is built using a type $\mathbf{t}$. Depending on whether $t \in \#_1$, $t \in \#_2$, $t \in \mathcal{S}_1(t')$ and $t \in \mathcal{S}_2(t'')$ (for some t' and t''), they are, at most, four possibilities for the two children states in a transition built using $\mathbf{t}$: $(\#_1, \#_2)$,$(\#_1, \mathcal{S}_2(t''))$, $(\mathcal{S}_1(t'), \#_2)$, $(\mathcal{S}_1(t'), \mathcal{S}_2(t''))$. 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 the parent state in a transition built using $\mathbf{t}$: $\mathcal{S}_1(t)$, $\mathcal{S}_2(t)$ and $\mathcal{F}$. The automaton has, at most, $3 \times 4 \times 2^n$ transitions.

4.4 Algorithm implementation

The algorithm works by computing transitions from pairs of known accessible states and marking the discovered states as accessible. The algorithm starts with the two states associated with leaves ($\#_1$ and $\#_2$) marked as accessible. Then, for each pair of accessible states we compute the set of all transitions using those states. Each new discovered state is marked as accessible.

The main operation of our algorithm is the computation of transitions. In order to compute it our prototype relies on a symbolic representation of types for fast enumerations. Once the formula is encoded into n formulae of the form $V_i = \langle a \rangle\, f_i(\omega_1, \ldots, \omega_n)$ or $V_i = P_j$ the complexity of the translation to an automaton is $O(n \times 2^n)$.

5 Experimental validation

First, we tested our translation with queries taken from the XPathMark benchmark [7]. Figure 1 presents the time spent in constructing the automaton for a given benchmark query. This demonstrates the practical feasibility of our automaton construction technique for practical query instances.

Second, we tested our prototype for solving the problem of XPath satisfiability in the presence of DTD constraints. This problem is known to be EXPTIME-complete for the vast majority of queries found in practice [2]. Specifically, the complexity depends both on the query size n and on the constraint (DTD) size m. As noticed in [16], a direct logical approach results in a $2^{\mathcal{O}(n \cdot m)}$ time complexity [10] whereas an automaton construction technique such as ours yields a better $\mathcal{O}(m) \cdot 2^{\mathcal{O}(n)}$ time complexity. This is because once an automaton is built from the query, it can then be simply intersected with the automaton representing the constraint. For this reason, state-of-the-art implementations of the direct logical technique (such as [10]) can hardly deal with recursive queries that require to unfold large DTDs (like XHTML). The implementation techniques we propose here extend the envelope of practically solvable problem instances, and speed up feasible cases considerably. For example, Figure 2 presents the overall time spent in solving the satisfiability problem for the XPath $U_n = (//tr/*)^n$ in the presence of two DTDs: XHTML basic and XHMTL strict. U_n tests if there

is a lineage of n elements "tr". Results illustrate the gain in using our prototype when compared to the solver of [10].

The prototype and an enriched version of this benchmark are available at the address: http://tyrex.inria.fr/a4mu.

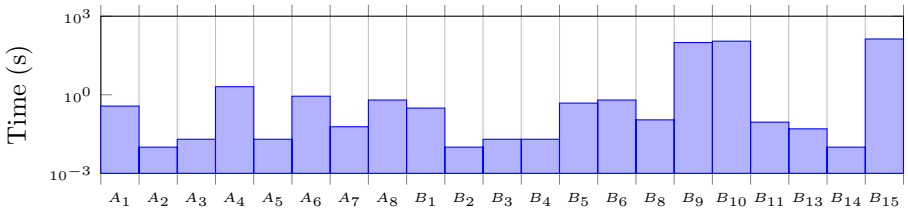

Fig. 1. Time spent in automaton construction for XPathMark (A_i) and (B_j) queries.

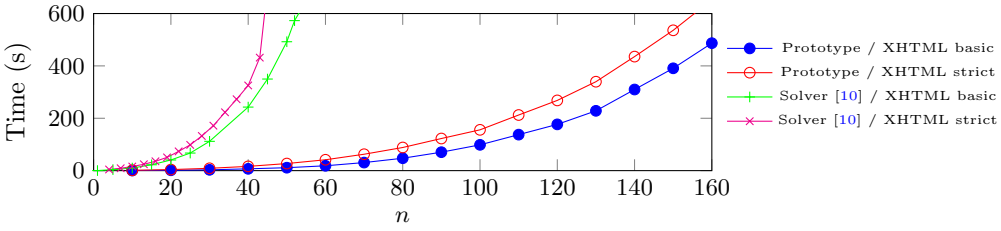

Fig. 2. Time spent in testing the satisfiability of U_n with XHTML strict or basic.

6 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 locally consistent tree annotations. We prove that local consistency of annotations correspond to their global 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 μ-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

1. Arroyuelo, D., Claude, F., Maneth, S., Mäkinen, V., Navarro, G., Nguyen, K., Sirén, J., Välimäki, N.: Fast in-memory XPath search using compressed indexes. Software: Practice and Experience 45(3) (2015)
2. Benedikt, M., Fan, W., Geerts, F.: XPath satisfiability in the presence of DTDs. In: PODS '05. ACM Press (2005)
3. Benzaken, V., Castagna, G., Nguyen, K., Siméon, J.: Static and dynamic semantics of NoSQL languages. In: POPL'13 (2013)
4. Calvanese, D., Giacomo, G.D., Lenzerini, M., Vardi, M.Y.: Node selection query languages for trees. In: AAAI (2010)
5. ten Cate, B., Litak, T., Marx, M.: Complete axiomatizations for XPath fragments. J. Applied Logic 8(2) (2010)
6. Doner, J.: Tree acceptors and some of their applications. Journal of Computer and System Sciences 4 (1970)
7. Franceschet, M.: XPathMark: An XPath benchmark for the XMark generated data. In: Bressan, S., Ceri, S., Hunt, E., Ives, Z., Bellahsène, Z., Rys, M., Unland, R. (eds.) Database and XML Technologies, Lecture Notes in Computer Science, vol. 3671. Springer Berlin Heidelberg (2005)
8. Francis, N., David, C., Libkin, L.: A direct translation from XPath to nondeterministic automata. In: Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management, Santiago, Chile. (2011)
9. Friedmann, O., Lange, M.: A solver for modal fixpoint logics. Electron. Notes Theor. Comput. Sci. 262 (May 2010)
10. Genevès, P., Layaïda, N., Schmitt, A.: Efficient static analysis of XML paths and types. In: PLDI '07. ACM Press, New York, NY, USA (2007)
11. Genevès, P., Layaïda, N., Schmitt, A., Gesbert, N.: Efficiently Deciding μ-calculus with Converse over Finite Trees. To appear in ACM Trans. Comput. Log. (2015)
12. Klarlund, N., Møller, A.: MONA Version 1.4 User Manual. BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus (January 2001)
13. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA implementation secrets. In: CIAA '00. LNCS, vol. 2088. Springer-Verlag, London, UK (2001)
14. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science 27 (1983)
15. Libkin, L., Sirangelo, C.: Reasoning about xml with temporal logics and automata. In: LPAR '08. Springer-Verlag, Berlin, Heidelberg (2008)
16. Libkin, L., Sirangelo, C.: Reasoning about XML with temporal logics and automata. J. Applied Logic 8(2) (2010)
17. Pan, G., Sattler, U., Vardi, M.Y.: BDD-based decision procedures for the modal logic K. Journal of Applied Non-classical Logics 16(1-2) (2006)
18. Tanabe, Y., Takahashi, K., Yamamoto, M., Tozawa, A., Hagiya, M.: A decision procedure for the alternation-free two-way modal μ-calculus. In: In TABLEAUX 2005. LNCS, vol. 3702. Springer-Verlag, London, UK (September 2005)
19. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory 2(1) (1968)
20. Ünel, G., Toman, D.: An incremental technique for automata-based decision procedures. In: CADE'07 (2007)