An Efficient Translation from Modal μ-Calculus with Converse to Tree Automata

Abstract

We present a direct translation from a sub-logic of μ-calculus to non-deterministic automata of finite binary 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). 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 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. (Pan, Sattler, and Vardi 2006) and (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 μ-calculus (without converse modalities) (Kozen 1983). In general, however, automata-based decision procedures implementations are often outperformed by alternative techniques such as tableau methods, as found in e.g. (Pan, Sattler, and Vardi 2006; Tanabe et al. 2005; Genevès et al. 2015). 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 (Libkin and Sirangelo 2010). 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 (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, queries 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 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 (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 in (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 automaton 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 time complexity for satisfiability is $2^{O(n)}$.

More recently (Libkin and Sirangelo 2008) introduced a translation from CXPath to tree automata in $2^{O(n)}$. This work was further developed in (Libkin and Sirangelo 2010) and (Francis, David, and Libkin 2011). 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).

In the meantime, (Calvanese et al. 2010) presented a two steps translation from μXPath formulas (a MSO-complete μ-calculus variant) to non-deterministic tree automata. The first step is to translate the formulas to two-way alternating tree automata and the second step is to translate these automata to non-deterministic tree automata. The size of the obtained tree automaton is $2^{O(n^2)}$. The second step has been improved (Björklund, Gelade, and Martens 2010) for a sublogic such that the intermediary two-way alternating tree automata are "loop-free". From this sublogic, a non deterministic tree automaton with $2^{O(n)}$ states can be constructed. More precisely, if n is the size of the lean (a subset of the Fischer-Ladner closure presented here) of the formula, their automaton has 2^{2n} states and 2^{4n} transitions. Their translation has no reported implementation.

Contributions. Our contributions are twofold. First, we present a new direct translation from $\mathcal{L}_\mu$ to tree automata. This translation produces an automaton that has, at most, 2^n transitions where n is the size of the lean. This improves on the best known translation that has 2^{4n} transitions. Second, our 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.

Logical context

Formulae

The logic we consider is a fragment of the alternation free modal μ-calculus presented below. The grammar of our formulae is given in figure 1.

In this paper we suppose that the formulae we are given are closed formulae and that each variable name is only bound once (eventually thanks to an α-conversion). Finally, we impose that formulae respect the "cycle-freeness" and the "guarded variables" properties of the language $\mathcal{L}_\mu$ that are presented later.

φ	$::=$		formula
	$\mid$	P	atomic proposition
	$\mid$	$f(\varphi_1, \ldots, \varphi_k)$	f is a boolean function
	$\mid$	X	variable
	$\mid$	$\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi$	polyadic fixpoint
	$\mid$	$\langle a \rangle\, \varphi$	modality

Figure 1: Grammar of formulae

Focused trees

We consider only finite binary trees (our results transfer to unranked trees thanks to the well-known "first-child next-sibling" bijective mapping: see e.g. (Genevès, Layaïda, and Schmitt 2007)). The models of our formulae are focused trees; focused trees are trees with the additional information of which node we are focused on. The set of all focused trees (and therefore the set of all possible models) is $\mathcal{F}$.

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).

Atomic propositions

Atomic propositions corresponds to propositions that can be verified directly against a node. We write $P \vdash \mathcal{T}$ when the proposition P is true on the node focused by $\mathcal{T}$.

Fixpoints

The formulae of the form $\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi$ in our grammar correspond to fixpoints. Each fixpoint has a finite number of variables indexed by a set I (each fixpoint can have a different number of variables and therefore a different I).

Boolean functions

When using the "natural" interpretation of formulae, the set of models for a formula $a \wedge b$ is the intersection of the set of models for the formula a and for the formula b. We present here how to transform a boolean function to a function over sets and extend this "natural" interpretation.

Let $f : \{0,1\}^k \to \{0,1\}$ be a boolean function of arity k and k sets of focused trees $S_1, \ldots, S_k$, we extend f to a set function by defining $f(S_1, \ldots, S_k)$ as $\{x \in \mathcal{F} \mid f(\chi(x, S_1), \ldots, \chi(x, S_k))\}$ with $\chi(x, E) = 1$ when $x \in E$ and $\chi(x, E) = 0$ otherwise.

For the simple functions $f_\wedge(\varphi_1, \varphi_2) = \varphi_1 \wedge \varphi_2$, $f_\vee(\varphi_1, \varphi_2) = \varphi_1 \vee \varphi_2$, $f_\neg(\varphi) = \neg\varphi$ and $f_\top() = \top$ we do have $f_\wedge(A, B) = A \cap B$, $f_\vee(A, B) = A \cup B$, $f_\neg(A) = \mathcal{F} \backslash B$ and $f_\top = \mathcal{F}$.

We restrict the use of boolean functions in formulae. In a formula $f(\varphi_1, \ldots, \varphi_k)$, we either have φ_i a closed formula or f use its i-th argument positively: $\forall (x_1, \ldots, x_k) \in \{0,1\}^k f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_k) \leq f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_k))$.

Interpretation of formulae

The interpretation of a formula φ is recursively defined as $[\![\varphi]\!]V$ where V is the environment mapping the free variables of φ to sets of models. Figure 2 presents the definition of $[\![\varphi]\!]V$. The notation $V[A \to B]$ indicates that the environment 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).

Lemma 1. *Due to the restrictions on boolean functions, the interpretation is growing in environment i.e.* $A \subseteq B \Rightarrow [\![\varphi]\!]V[X \to A] \subseteq [\![\varphi]\!]V[X \to B]$.

$$
\begin{array}{rcl}
[\![X]\!]V & = & V(X) \\
[\![P]\!]V & = & \{\, \mathcal{T} \in \mathcal{F} \mid P \vdash \mathcal{T}\} \\
[\![f(\varphi_1, \ldots, \varphi_n)]\!]V & = & f\big([\![\varphi_1]\!]V, \ldots, [\![\varphi_n]\!]V\big) \\
[\![\langle a \rangle\, \varphi]\!]V & = & \{\mathcal{T}\,\langle a \rangle \mid \mathcal{T} \in [\![\varphi]\!]V \\
 & & \wedge \mathcal{T}\,\langle a \rangle \text{ is defined}\} \\
[\![\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi]\!]V & = & [\![\psi]\!]V[X_i \to U_i]_{i \in I}
\end{array}
$$

With $U_j = \bigcap_{(T_1,\ldots,T_n) \in S} T_j$ and $S = \{(T_1, \ldots, T_n) \in \mathcal{P}(\mathcal{F})^n \mid \forall j([\![\varphi_j]\!]V[X_i \to T_i]_{i \in I} \subseteq T_j)\}$.

Figure 2: Interpretation of formulae

Guarded variables

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. The first restriction we impose on formulae is **for every fixpoint $\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi$, and for $i \in I$ the variables $(X_j)_{j \in I}$ have to be guarded in φ_i.** This restriction is used to transform a formula with unguarded fixpoints to a formula where all fixpoints are guarded; we expand the unguarded fixpoints (i.e. we replace variables by their definition) and since variables are guarded the resulting formula has guarded fixpoints.

Definition 1. *The* unfolding *of a formula φ is written as $\mathit{unf}(\varphi)$ and is defined as:*

$$
\begin{array}{rcl}
\mathit{unf}(P)) & = & P \\
\mathit{unf}(f(\varphi_1, \ldots, \varphi_k)) & = & f(\mathit{unf}(\varphi_1), \ldots, \mathit{unf}(\varphi_k)) \\
\mathit{unf}(\langle a \rangle\, \varphi) & = & \langle a \rangle\, \varphi \\
\mathit{unf}(\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi) & = & \mathit{unf}(\psi')
\end{array}
$$

where $\psi' = \psi\{X_j/\mu(X_i = \varphi_i)_{i \in I}\ in\ \varphi_j\}_{j \in I}$ is the formula where for $j \in I$ the occurrences of the variables X_j are replaced by $\mu(X_i = \varphi_i)_{i \in I}\ in\ X_j$.

Lemma 2. *For a closed formula φ, $\mathit{unf}(\varphi)$ is well-defined, in $\mathit{unf}(\varphi)$ all fixpoints are guarded and $\mathit{unf}(\varphi)$ has the same semantic as φ (i.e. $[\![\varphi]\!]V = [\![\mathit{unf}(\varphi)]\!]V$).*

Proof. The whole proof is shown in the appendix.

The main idea of the proof is to use the order on formulae: first on the set of unguarded fixpoints and then on the size of the formula. After that, the harder part is proving that expanding a fixpoint does not change its semantic.

Let $U_j = \bigcap_{(T_1,\ldots,T_n) \in S} T_j$ and $S = \{(T_1, \ldots, T_n) \in \mathcal{P}(\mathcal{F})^n \mid \forall j([\![\varphi_j]\!]V[X_i \to T_i]_{i \in I} \subseteq T_j)\}$, we have that $U_j = [\![\varphi_j]\!]V[X_j \to U_j]$ because the function $(T_i)_{i \in I} \to ([\![\varphi_i]\!]V[X_j \to T_j])$ is a growing function.

If X is a variable and Y a formula, then $[\![\psi\{X/Y\}]\!]V = [\![\psi]\!]V[X \to [\![Y]\!]V]$ and thus $[\![\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi]\!]V = [\![\psi]\!]V[X_i \to jU_j] = [\![\psi]\!]V[X_i \to [\![\varphi_j]\!]V[X_j \to U_j]] = [\![\psi\{X_i/\mu(X_i = \varphi_i)_{i \in I}\ in\ \varphi_j\}]\!]V$.

□

Cycle-free formulae

Definition 2. *Paths are suits of programs. 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:*

$$
\begin{array}{rcl}
\mathbb{P}(P) & = & \emptyset \\
\mathbb{P}(f(\varphi_1, \ldots, \varphi_n)) & = & \bigcup_{i=1}^{n} \mathbb{P}(\varphi_i) \\
\mathbb{P}(\langle a \rangle\, \varphi) & = & \{\langle a \rangle\} \cup \{\langle a \rangle\, p \mid p \in \mathbb{P}(\mathit{unf}(\varphi))\}
\end{array}
$$

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

Definition 3. *Let φ be a cycle-free formula and $\mathcal{T}$ be a fixed focused binary tree, we take the smallest c such that no path of φ has c cycles and we define $c(\varphi, \mathcal{T}) = c \times |\mathcal{T}|$.*

Lemma 3. *Any path of φ that is longer than $c(\varphi, \mathcal{T})$ 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. □

$\mathcal{L}$ean, type, consistent type

Definition 4. *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:*

$$
\begin{array}{rcl}
\mathcal{L}ean\,(P) & = & \{P\} \\
\mathcal{L}ean\,(f(\varphi_1, \ldots, \varphi_k)) & = & \cup_{i=1}^{k} \mathcal{L}ean\,(\varphi_i) \\
\mathcal{L}ean\,(\langle a \rangle\, \varphi) & = & \{\langle a \rangle\, \varphi\} \cup \mathcal{L}ean\,(\mathit{unf}(\varphi))
\end{array}
$$

Lemma 4. *For a formula ψ, $|\mathcal{L}ean(\mathit{unf}(\psi))| = O(|\psi|)$.*

Proof. The set $\mathcal{L}\text{ean}(\text{unf}(\psi))$ 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. $|\mathcal{L}\text{ean}(\psi)|$ is at most linear in the size of ψ.

□

We consider a formula ξ and we write $\mathcal{L}$ean for $\mathcal{L}\text{ean}(\text{unf}(\xi))$, n for the size of $\mathcal{L}\text{ean}(\text{unf}())$ and $\omega_1, \ldots, \omega_n$ for $\mathcal{L}$ean. A formula φ for which $\mathcal{L}\text{ean}(\text{unf}(\varphi))$ is included in $\mathcal{L}$ean is called a $\mathcal{L}$ean formula. First, we need to introduce the lemmas 5 and 6.

Lemma 5. *For a $\mathcal{L}$ean-formula φ there is a boolean function f_φ such that $[\![\varphi]\!] = [\![f_\varphi(\omega_1, \ldots, \omega_n)]\!]$.*

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

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 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 $\mathrm{unf}(\kappa) = f(\omega_1, \ldots, \omega_n)$. □

Lemma 6. *For $i \in 1..n$ we have either an atomic proposition P such that $[\![\omega_i]\!]V = [\![P]\!]V$ or it exists a_i and f_i such that $[\![\omega_i]\!]V = [\![\langle a_i \rangle f_i(\omega_1, \ldots, \omega_n)]\!]V$.*

Proof. By definition of the $\mathcal{L}$ean, for $i \in 1..n$ we have either $\omega_i = P$ or $\omega_i = \langle a_i \rangle \varphi_i$ for some P, a_i and φ_i. Which means we either have $[\![\omega_i]\!]V = [\![P]\!]V$ or it exists f_i such that $[\![\varphi_i]\!]V = [\![f_i(\omega_1, \ldots, \omega_n)]\!]V$ and thus $[\![\omega_i]\!]V = [\![\langle a_i \rangle f_i(\omega_1, \ldots, \omega_n)]\!]V$. □

Remark 1. *The $(\omega_i)_{i \in 1..n}$ represent the $\mathcal{L}$ean as a set and are, by definition, all different syntactically. We can have semantically equivalent but syntactically different formulae (like $\omega_1 = a \vee b$ and $\omega_2 = b \vee a$) but this is only possible for modal formula (of the form $\omega_i = \langle a_i \rangle \varphi_i$): for $\omega_i = P$ any equivalent formula should be $\omega_j = P'$ with $[\![P]\!]V = [\![P']\!]V$ thus $P = P'$. It is clear that a ω_i cannot be semantically equivalent to a modal formula and an atomic proposition.*

Definition 5. *A* type *is an element of $\{0,1\}^n$. For a type $\mathbf{t}$ and a boolean function f of arity n, we write $f(\mathbf{t})$ for $f(\mathbf{t}_1, \ldots, \mathbf{t}_1)$. A type $\mathbf{t}$ is said consistent with a focused tree $\mathcal{T}$ when $\forall i \in 1..n$ $(\mathbf{t}_1 = 1 \Leftrightarrow \mathcal{T} \in [\![\omega_i]\!])$.*

We denote by $\mathcal{L}$ the set of i such that ω_i represents an atomic proposition; F_a the set of ω_i using the modality a, $F_a = \left\{ i \mid \left([\![\omega_i]\!]V = [\![\langle a \rangle f_i(\omega_1, \ldots, \omega_n)]\!]V \right) \in \mathcal{L}\mathrm{ean} \right\}$.

Definition 6. *Let $\widetilde{1} = \bar{2}$ and $\widetilde{2} = \bar{1}$, let $\mathbf{y}$ be a type and $a \in \{1,2\}$, the set $S_a(\mathbf{y})$ of types compatible with being the a-th parent of $\mathbf{y}$ is $S_a(\mathbf{y}) = \left\{ x \in \{0,1\}^n \mid \forall i \in F_{\widetilde{a}}\ \mathbf{y}_i = 0 \wedge \begin{smallmatrix} \forall i \in F_a\ x_i = f_i(\mathbf{y}) \\ \forall i \in F_{\bar{a}}\ \mathbf{y}_i = f_i(x) \end{smallmatrix} \right\}$.*

Definition 7. *For $i \in \{1,2\}$, $\#_i$ is the set of types having no i-child: $\#_i = \{x \mid \forall v \in F_i\ x_v = 0\}$.*

Lemma 7. *S_a is such that $S_a(x) \cap S_a(y) \neq \emptyset$ then we necessarily have $S_a(x) = S_a(y)$.*

Proof. Let y and x be two types, $a \in \{1,2\}$, let $z \in S_a(x) \cap S_a(y)$ and $w \in S_a(x)$. We have $\forall i \in F_a f_i(y) = z_i = f_i(x) = w_i$, $\forall i \in F_{\bar{a}} y_i = f_i(z) = x_i = f_i(w)$ and finally $\forall i \in F_{\widetilde{a}}\ y_i = x_i = 0$ therefore $w \in S_a(y)$. □

Definition 8. *The set of atomic propositions appearing in ξ is $\mathcal{L}$. A type t forces the value of each proposition in $\mathcal{L}$. We write $\mathcal{L}(t)$ for the set of atomic proposition implied by t.*

Annotation of trees

We introduce the notions of locally and globally consistent tree annotations and prove that they are equivalent. Local consistency can be used to derive an automaton, 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 $\gamma(\mathcal{N})$. $\gamma(\mathcal{N})_i$ denotes the i-th component of $\gamma(\mathcal{N})$ (if $\gamma(\mathcal{N}) = (t_1, \ldots, t_n)$ then $\gamma(\mathcal{N})_i = t_i$). 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 $\mathcal{N}$, $\{P \in \mathcal{L} \mid P \vdash \mathcal{N}\} = \mathcal{L}(\gamma(\mathcal{N}))$;*
- *if $\mathcal{N}_c$ is the a-child of $\mathcal{N}_p$, then $\gamma(N_p) \in S_a(\gamma(N_c))$;*
- *if $\mathcal{N}$ has no a-child $\gamma(\mathcal{N}) \in \#_a$;*
- *if $\mathcal{N}$ has no parent then there are no $i \in F_{\bar{1}} \cup F_{\bar{2}}$ such that $\gamma(\mathcal{N})_i = 1$.*

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

Proof. The idea of the proof (see appendix) relies on a notion of consistency at a distance k. We show that local consistency is consistency at distance k for all k. Then we show that for k big enough, the consistency at distance k is the consistency.

Given a formula φ and a tree $\mathcal{T}$, all paths of $\mathbb{P}(\varphi)$ are valid on $\mathcal{T}$ if they have less than $c(\mathcal{T}, \varphi)$ programs. Thus, we only need to show that the local consistency checks the consistency on all finite paths. □

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 would lead to a bottom-up non-deterministic tree automaton that has 2^n states and $(2^n)^3$ transitions. We introduce here the notion of interface that allows a bottom-up non-deterministic tree automaton that is much smaller ($O(2^n)$ transitions in the worst case) and more prone to optimization.

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 11. *$\mathcal{F}$ is the set of types that are compatible with being a root and a solution of ξ.*

$$\mathcal{F} = \left\{ t \in \{0,1\}^n \mid f_\xi(t) = 1 \wedge (\forall i \in (F_{\bar{1}} \cup F_{\bar{2}}), t_i = 0) \right\}$$

Definition 12. *Let $C_i = \{\mathcal{S}_i(\mathbf{t}) \mid \mathbf{t}$ type$\} \cup \#_i$, the set of states of our automaton is $\mathcal{Q} = (C_1 \times \{1\}) \cup (C_2 \times \{2\}) \cup \{(\mathcal{F}, 0)\}$.*

Remark 2. *As a consequence of remark 7, 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 $\mathcal{S}_i(y_1) = \mathcal{S}_i(y_2)$.*

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 $\forall p \in \mathcal{L}\ (p \in l \Leftrightarrow n \in [\![p]\!])$

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 8. *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 $\tau(n) \in \mathcal{S}_i(\tau(n_i))$. Let n be a node, if n has no i-child, $\tau(n) \in \#_i$ which implies $\tau(n)_i = 0$ for $i \in F_i$. Finally, the root r is associated with $\tau(r) \in \mathcal{F}$ and therefore $\tau(r)_i = 0$ 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. □

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.

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 $\omega_i = \langle a \rangle f_i(\omega_1, \ldots, \omega_n)$ or $\omega_i = P$ the complexity of the translation to an automaton is $O(n \times 2^n)$.

Experimental validation

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

Typing XPath

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.

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 statis 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: 111.41 0.09, 0.18, 0.05, 0.01 and 135.05 seconds, respectively. These queries translate into either empty or very small automata, because their structure is simple.

Table 3 presents for each query the time spent (in seconds), the $\mathcal{L}$ean size, the lg (log in base 2) of the number of states and the lg of the number of transitions of the resulting automaton. Table 3 shows that for most queries, the translation is done in less than a second and the size of the resulting automaton is much smaller than the worst case $O(2^n)$.

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	1.01	0.06	0.67	0.52	0.01	0.02	0.02	0.63	0.66	0.06	0.12	94.8
Lean size	22	13	17	25	18	23	19	23	23	16	17	17	24	23	17	22	42
lg(#states)	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(#trans)	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 3: Statistics of the translation of the XPathMark Benchmark

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 $\neg e \wedge e'$ or by intersecting the automata for e' and $\neg e$.

To benchmark our prototype we re-used the containment problems from the paper that introduced the btl-solver (Genevès et al. 2015). Table 4 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 $\neg e \wedge e'$ to automata and testing it for emptyness *(method full-$a4\mu$)* and testing the satisfiability with the btl-solver.

Unsurprisingly the method *full-$a4\mu$* is outperformed by the method *inter-$a4\mu$*. 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 terminates quickly (for instance in $e5 \subseteq e6$).

Problem	Answer	inter-$a4\mu$	full-$a4\mu$	btl-solver
$e1 \subseteq e2$	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

Figure 4: Time in seconds for $a4\mu$ and the btl-solver

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 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. For this reason, state-of-the-art implementations of the direct logical technique can hardly deal with recursive queries that require to unfold large schema (like XHTML). Table 5 shows that the implementation techniques we propose here extend the envelope of practically solvable problem instances, and speed up feasible cases considerably. Queries $e8$, $e9$, $e10$ and $e11$ are also taken from the benchmark of the btl-solver, with $e13 = e10 \cup e11 \cup e12$.

Figure 6 reports on satisfiability-testing times for the XPath $(//tr/*)^n$ under the XHTML Strict and Basic DTDs.

Problem	DTD	Answer	inter-$a4\mu$	btl-solver
$e8$	$None$	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 5: Time in seconds for $a4\mu$ and the btl-solver

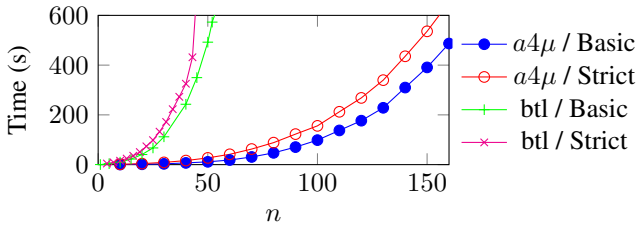

Figure 6: Satisfiability of $(//tr/*)^n$ under constraints.

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

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

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.

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

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

Franceschet, M. 2005. XPathMark: An XPath benchmark for the XMark generated data. In Bressan, S.; Ceri, S.; Hunt, E.; Ives, Z.; Bellahsène, Z.; Rys, M.; and Unland, R., eds., *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 *Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management, Santiago, Chile.*

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

Genevès, P.; Layaïda, N.; Schmitt, A.; and Gesbert, N. 2015. Efficiently Deciding μ-calculus with Converse over Finite Trees. *To appear in 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 μ-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).

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).

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 μ-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*.

Proof of lemmas

Lemma (1). *Due to the restrictions on boolean functions, the interpretation is growing in environment i.e.* $A \subseteq B \Rightarrow [\![\varphi]\!]V[Y \to A] \subseteq [\![\varphi]\!]V[Y \to B]$.

Proof. Using the recursive order used to define $[\![\varphi]\!]V$ we have:

- $[\![X]\!]V[Y \to A] = A \subseteq B = [\![X]\!]V[X \to B]$ for $X = Y$;
- $[\![X]\!]V[Y \to A] = V(X) = [\![X]\!]V[Y \to B]$ for $X \neq Y$;
- $[\![P]\!]V[Y \to A] = [\![P]\!]V[Y \to B]$;
- for a formula $[\![f(\varphi_1, \ldots, \varphi_n)]\!]V$, for all closed formulae φ_j we have $[\![\varphi_j]\!]V[Y \to A] = [\![\varphi_j]\!]V[Y \to B] = [\![\varphi_j]\!]\emptyset]$ and for φ_i an open formula, we recursively have $[\![\varphi_j]\!]V[Y \to A] \subseteq [\![\varphi_j]\!]V[Y \to B]$ and f use its j-th argument positively which means $[\![f(\varphi_1, \ldots, \varphi_n)]\!]V[Y \to A] = f([\![\varphi_1]\!]V[Y \to A], \ldots, [\![\varphi_n V[Y \to A]]\!]) \subseteq f([\![\varphi_1]\!]V[Y \to A], \ldots, [\![\varphi_j]\!]V[Y \to B], \ldots, [\![\varphi_n V[Y \to A]]\!])$;
- $[\![\langle a \rangle\, \varphi]\!]V[Y \to A] \subseteq [\![\langle a \rangle\, \varphi]\!]V[Y \to B]$ since $[\![\varphi]\!]V[Y \to A] \subseteq [\![\varphi]\!]V[Y \to B]$;
- $[\![\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi]\!]V[Y \to C] = [\![\psi]\!]V[Y \to C, X_i \to U_i]_{i \in I}$, $U_j = \bigcap_{(T_1, \ldots, T_n) \in S} T_j$ and $S = \{(T_1, \ldots, T_n) \in \mathcal{P}(\mathcal{F})^n \mid \forall j([\![\varphi_j]\!]V[Y \to C, X_i \to T_i]_{i \in I} \subseteq T_j)\}$; the sets $[\![\varphi_j]\!]V[Y \to C, X_i \to T_i]_{i \in I}$ are bigger when $C = B$ than when $C = A$ (by the recursive hypothesis) which means S is smaller and therefore the U_j are larger. The U_i being larger in the case $C = B$, recursively we have $[\![\psi]\!]V[X_i \to U_i]$ that is also larger.

□

Lemma (2). *For a closed formula* φ*,* unf(φ) *is well-defined, in* unf(φ) *all fixpoints are guarded.*

Proof. We introduce an order on formulae: first on the set of unguarded fixpoints and then on the size of the formula.

We will prove that the only recursive uses of unf(ψ) in the definition of unf(φ) are with a ψ smaller in the sense of definition .

- For $\varphi = P$ and $\varphi = \langle a \rangle\, \psi$, unf$(\varphi)$ does not use recursion therefore it is well-defined.
- For $\varphi = f(\varphi_1, \ldots, \varphi_k)$, each φ_i has a subset of the free fixpoints of φ and is of smaller size. Thus the φ_i are smaller in the sense of definition .
- For $\varphi = \mu(X_i = \varphi_i)_{i \in I}\ in\ \psi$, all occurrences of the X_i are guarded (cf restriction **??**) therefore the expansion into $\psi\{X_i/\mu(X_i = \varphi_i)_{i \in I}\ in\ X_i\}$ removes the unguarded fixpoint φ.

Using the order used to prove the well-definedness of unf(φ) we have:

- P and $\langle a \rangle\, \psi$ have no unguarded fixpoint
- recursively the unf(φ_i) have no unguarded fixpoint therefore unf$(f(\varphi_1, \ldots, \varphi_k))$ has none either.
- unf$(\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi) =$ unf$(\psi\{X_i/\mu(X_i = \varphi_i)_{i \in I}\ in\ X - I\})$ and unf$(\psi\{X_i/\mu(X_i = \varphi_i)_{i \in I}\ in\ X_i\})$ has no unguarded fixpoint.

□

Lemma (2 bis). *unf*(φ) *has the same semantic as* φ *(i.e.* $[\![\varphi]\!]V = [\![$*unf*$(\varphi)]\!]V$*).*

Proof. The only peculiar point of this proof lies in proving that expanding a fixpoint does not change its semantic, or $[\![\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi]\!]V = [\![\psi\{X_i/\mu(X_i = \varphi_i)_{i \in I}\ in\ X_i\}]\!]V$.

We write $(T_i)_{i \in I} \dot{\subseteq} (N_i)_{i \in I}$ for $\forall\ i \in I\ (T_i \subseteq N_i)$ and $(T_i)_{i \in I} \dot{\cap} (N_i)_{i \in I}$ for $(T_i \cap N_i)_{i \in I}$. For any x, y we have $x \dot{\cap} y \dot{\subseteq} x$.

Let $f((T_j)_{j \in I}) = \left([\![\varphi_j]\!]V[X_i \to T_i]\right)_{j \in I}$ by componentwise applications of lemma 1 we have $(T_i)_{i \in I} \dot{\subseteq} (N_i)_{i \in I} \Rightarrow f((T_i)_{i \in I}) \dot{\subseteq} f((N_i)_{i \in I})$.

Let $(U_j)_{j \in I} = \dot{\bigcap}_{x \in S} x$ and $S = \{x \in \mathcal{P}(\mathcal{F})^I \mid (f(x) \dot{\subseteq} x)\}$, we have $f((U_j)_{j \in I}) = f(\dot{\bigcap}_{x \in S} x) \subseteq \dot{\bigcap}_{x \in S} f(x) \subseteq \dot{\bigcap}_{x \in S} x = (U_j)_{j \in I}$ but $f((U_j)_{j \in I}) \dot{\subseteq} (U_j)_{j \in I} \Rightarrow f(f((U_j)_{j \in I})) \dot{\subseteq} f((U_j)_{j \in I})$ therefore $f((U_j)_{j \in I}) \in S$ and thus $(U_j)_{j \in I} \dot{\subseteq} f((U_j)_{j \in I})$ which gives us $(U_j)_{j \in I} = f((U_j)_{j \in I}) = ([\![\varphi_j]\!]V[X_j \to U_j])_{j \in I}$

If X is a variable and Y a formula, then $[\![\psi\{X/Y\}]\!]V = [\![\psi]\!]V[X \to [\![Y]\!]V]$ and thus $[\![\mu(X_i = \varphi_i)_{i \in I}\ in\ \psi]\!] = [\![\psi]\!]V[X_i \to jU_j] = [\![\psi]\!]V[X_i \to [\![\varphi_j]\!]V[X_j \to U_j]] = [\![\psi\{X_i/\mu(X_i = \varphi_i)_{i \in I}\ in\ \varphi_j\}]\!]V$.

□

Proof of the theorem

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 13. *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* $\varphi = P$ *there is a unique* i *such that* $[\![\omega_i]\!] = [\![P]\!]$, $\mathcal{V}_\gamma(\varphi, \mathcal{T}, k) = (\gamma(t)_i)$
- $\mathcal{V}_\gamma(f(\varphi_1, \ldots, \varphi_n), \mathcal{T}, k) = f(\mathcal{V}_\gamma(\varphi_1, \mathcal{T}, k), \ldots, \mathcal{V}_\gamma(\varphi_n, \mathcal{T}, k))$
- $$\mathcal{V}_\gamma(\langle a \rangle\, \varphi, \mathcal{T}, k) =$$

$$\begin{cases} \gamma(t)_i & \text{if } k = 0 \\ i \text{ such that} & \omega_i = \langle a \rangle \varphi \\ \\ \mathcal{V}_\gamma(\text{unf}(\varphi), \mathcal{T}\langle a \rangle, k-1) & \text{if } k > 0 \wedge \mathcal{T}\langle a \rangle \text{ exists} \\ \\ 0 & \text{if } k > 0 \\ & \wedge \mathcal{T}\langle a \rangle \text{ does not exist} \end{cases}$$

Lemma 9. *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.

- $\mathcal{V}_\gamma(P_j, \mathcal{T}, 1) = \gamma(t)_i = \mathcal{V}_\gamma(P_j, \mathcal{T}, 0)$ (for the i such that $[\![\omega_i]\!] = [\![P_j]\!]$) ;
- $$\mathcal{V}_\gamma(f(\varphi_1, \ldots, \varphi_k), \mathcal{T}, 1) = f(\mathcal{V}_\gamma(\varphi_1, \mathcal{T}, 1), \ldots, \mathcal{V}_\gamma(\varphi_k, \mathcal{T}, 1))$$
$$= f(\mathcal{V}_\gamma(\varphi_1, \mathcal{T}, 0), \ldots, \mathcal{V}_\gamma(\varphi_k, \mathcal{T}, 0))$$
$$= \mathcal{V}_\gamma(f(\varphi_1, \ldots, \varphi_k), \mathcal{T}, 0)$$
- $\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 1) =$
$$\begin{cases} \mathcal{V}_\gamma(\text{unf}(\varphi), \mathcal{T}\langle a \rangle, k-1) & \text{when } \mathcal{T}\langle a \rangle \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$
Let $i \in F_a$ be such that $\omega_i = \langle a \rangle \varphi$, 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 $\gamma(\mathcal{T})_i = 0$ and $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 1) = 0 = \gamma(\mathcal{T})_i = \mathcal{V}_\gamma(\varphi, \mathcal{T}, 0)$.
If $\mathcal{T}\langle a \rangle$ does exist:
$$\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 1) = \mathcal{V}_\gamma(\text{unf}(\varphi), \mathcal{T}\langle a \rangle, 0) =$$
$$f_i\left(\mathcal{V}_\gamma(\omega_1, \mathcal{T}\langle a \rangle, 0), \ldots, \mathcal{V}_\gamma(\omega_n, \mathcal{T}\langle a \rangle, 0)\right)) =$$
$$f_i(\gamma(T\langle a \rangle)_1, \ldots, \gamma(T\langle a \rangle)_1) =$$
$$f_i(\gamma(\mathcal{T}\langle a \rangle)) = \gamma(\mathcal{T})_i = \mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, 0)$$
The equality $f_i(\gamma(\mathcal{T}\langle a \rangle)) = \gamma(\mathcal{T})_i$ stands by definition of $\gamma(\mathcal{T}) \in \gamma(\mathcal{T}\langle a \rangle))$.

□

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

Lemma 10. *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 $\mathbb{P}(\varphi)$ longer than k are not valid paths of $\mathcal{T}$ then $\mathcal{V}_\gamma(\varphi, \mathcal{T}, k) = \chi(\mathcal{T}, [\![\varphi]\!])$.

- $\mathcal{V}_\gamma(P_j, \mathcal{T}, k) = \gamma(t)_i = (\mathcal{T} \in [\![\varphi]\!])$ (for i such that $[\![\omega_i]\!] = [\![P_j]\!]$);
- $\mathcal{V}_\gamma(f(\varphi_1, \ldots, \varphi_k), \mathcal{T}, k) = f(\mathcal{V}_\gamma(\varphi_1, \mathcal{T}, k), \ldots, \mathcal{V}_\gamma(\varphi_k, \mathcal{T}, k)) = f(\chi(\varphi_1, \mathcal{T}, k), \ldots, \chi(\varphi_k, \mathcal{T}, k)) = \chi(\mathcal{T}, [\![f(\varphi_1, \ldots, \varphi_k)]\!])$;
- 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
$\mathcal{V}_\gamma(\langle a \rangle \varphi, \mathcal{T}, k) = \mathcal{V}_\gamma(\text{unf}(\varphi), \mathcal{T}\langle a \rangle, k-1)$ and
$\mathcal{V}_\gamma(\text{unf}(\varphi), \langle a \rangle \mathcal{T}, k-1) = \chi(\mathcal{T}\langle a \rangle, [\![\text{unf}(\varphi)]\!]) = \chi(\mathcal{T}\langle a \rangle, [\![\varphi]\!]) = \chi(\mathcal{T}\langle a \rangle \langle \bar{a} \rangle, [\![\langle a \rangle \varphi]\!]) = \chi(\mathcal{T}, [\![\langle a \rangle \varphi]\!])$. The constraint on paths holds, a path of $\mathbb{P}(\text{unf}(\varphi))$ on $\mathcal{T}\langle a \rangle$ of size k is a valid path of $\mathbb{P}(\varphi)$ 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 $p \in \mathbb{P}(\varphi)$ such that p is of size greater than $c(\varphi, \mathcal{T})$. Thus $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 0) = \mathcal{V}_\gamma(\varphi, \mathcal{T}, c(\varphi, \mathcal{T}) + 1) \Leftrightarrow \mathcal{T} \in [\![\varphi]\!]$ so $\mathcal{V}_\gamma(\varphi, \mathcal{T}, 0) \Leftrightarrow \mathcal{T} \in [\![\varphi]\!]$.

□