Type Synthesis for the Logical Solver: an Approach based on Query Automata

Louis Jachiet, Pierre Genevès, Nabil Layaïda

July 25, 2013

1 Introduction

The logical solver used in [4] currently looks for a finite tree over which the logical formula is satisfied. As described in [1], we propose to study how to compute the set of all satisfying trees. One difficulty is to build a workable yet succinct representation for the synthesized type from the logical model of types and paths. In particular, recursion should be captured and backward axes may need to be rewritten in the forward direction if we stick to the usual representation of tree grammars, such as the ones used in CDuce.

In this document, we report on a logical based type synthesis, which paves the way for the research on polymorphism from a logical perspective. Specifically, we present a method and technique to build the set of satisfying trees for a given formula, and represent it as a forward-only tree-grammar based description. This technique relies on building query automata extracted from the XPath expressions given as input.

Outline

We first present basic preliminary definitions concerning formulas and types in Section 2. We give an automata perspective in Section 3. We prove the equivalence between the input formula and the obtained automaton in Section 4. We detail the containment for path formulas in Section 5. Finally we present the practical algorithm in Section 6 before we discuss experimental results and conclude in Section 7.

This work was supported by the ANR project TYPEX, ANR-11-BS02-007.

2 Preliminary Definitions

We first review the syntax of considered logical formulas, and describe their formal semantics.

2.1 Syntax

The syntax of the formulas over an alphabet of labels $\mathcal{A}$ and a disjointed alphabet of attributes $\mathcal{B}$ used is:

- $\varphi = \varphi_1 \vee \varphi_2$
- $\varphi = \varphi_1 \wedge \varphi_2$
- $\varphi = \neg\varphi'$
- $\varphi = \langle a \rangle \varphi'$ where $a \in \{1, 2, \bar{1}, \bar{2}\}$. We have $\bar{\bar{a}} = a$.
- $\varphi = \mu\overline{X_i = \varphi'_i}\ in\ \psi$
- X where X is a variable.
- $\varphi = \top$
- $\varphi = \sigma, \sigma \in \mathcal{A} \cup \mathcal{B}$

  The last two kind of formulas are called context-formulas.

2.2 Semantics

The trees we consider are over the alphabet $\mathcal{A}$, each node n has a label $\mathcal{L}(n) \in \mathcal{A}$, it also has a set $\mathcal{S}(n) \subset \mathcal{B}$ of attributes. We use the classic semantics:

- $[\![\varphi_1 \wedge \varphi_2]\!]_V = [\![\varphi_1]\!]_V \cap [\![\varphi_2]\!]_V$.
- $[\![\varphi_1 \vee \varphi_2]\!]_V = [\![\varphi_1]\!]_V \cup [\![\varphi_2]\!]_V$.
- $[\![\neg\varphi]\!]_V = \mathcal{F} \setminus [\![\varphi]\!]_V$.
- $[\![\langle a \rangle \varphi]\!]_V = \{\mathcal{T}\langle \bar{a} \rangle \mid \mathcal{T} \in [\![\varphi]\!]_V \wedge \mathcal{T}\langle \bar{a} \rangle \text{ defined }\}$
- $[\![\mu\overline{X_i = \varphi_i}\ in\ \psi]\!]_V = \text{let } T_i = \{\bigcap_{T_i \subset \mathcal{F}} T_i \mid [\![\varphi_i]\!]_{V\overline{\{T_i/X_i\}}} \subset T_i\} \text{ in } [\![\psi]\!]_{V\{T_i/X_i\}}$
- If $\varphi = \sigma$ where $\sigma \in \mathcal{A}$, $[\![\varphi]\!]_V = \{\mathcal{T} \in \mathcal{F}, \mathcal{L}(\mathcal{T}) = \sigma\}$.
- If $\varphi = \sigma$ where $\sigma \in \mathcal{B}$, $[\![\varphi]\!]_V = \{\mathcal{T} \in \mathcal{F}, \sigma \in \mathcal{S}(\mathcal{T})\}$.

2.3 Expanding fixpoints

Definition 1. *A fix point can be expanded by* $exp(\mu\overline{X_i = \varphi_i} \textit{ in } \psi) = \psi_{\mu\overline{X_i=\varphi_i} \textit{ in } \varphi_i / X_i}$.
The unfolding of a formula is the smallest set verifying:

- $unf(\varphi_1 \wedge \varphi_2) = \{u_1 \wedge u_2 \mid u_1 \in unf(\varphi_1), u_2 \in unf(\varphi_2)\}$
- $unf(\varphi_1 \vee \varphi_2) = \{u_1 \vee u_2 \mid u_1 \in unf(\varphi_1), u_2 \in unf(\varphi_2)\}$
- $unf(\neg\varphi) = \{\neg u \mid u \in unf(\varphi)\}$
- $unf(\langle a \rangle \varphi) = \{\langle a \rangle u \mid u \in unf(\varphi)\}$
- $unf(\sigma) = \{\sigma\}$ *for* σ *context-formula*
- $unf(\mu\overline{X_i = \varphi_i} \textit{ in } \psi) = unf(exp(\mu\overline{X_i = \varphi_i} \textit{ in } \psi)) \cup \{\mu\overline{X_i = \varphi_i} \textit{ in } \psi\}$

Remark 1. *It has been proved in [3] that exp does not change the semantics of a formula.*

Remark 2. *As we always unfold fix points we encounter, we don't have to consider variables as a case for induction on formulas.*

2.4 Modality-free variables

Definition 2. *We defined the set* $\mathcal{U}(\varphi)$ *of modality-free variables in* φ *as the smallest set verifying:*

- $\mathcal{U}(\varphi_1 \wedge \varphi_2) = \mathcal{U}(\varphi_1) \cup \mathcal{U}(\varphi_2)$
- $\mathcal{U}(\varphi_1 \vee \varphi_2) = \mathcal{U}(\varphi_1) \cup \mathcal{U}(\varphi_2)$
- $\mathcal{U}(\neg\varphi) = \mathcal{U}(\varphi)$
- $\mathcal{U}(\langle a \rangle \varphi) = \emptyset$
- $\mathcal{U}(\mu\overline{X_i = \varphi_i} \textit{ in } \psi) = \{X_i\} \sqcup \mathcal{U}(\psi)$
- $\mathcal{U}(\varphi) = \emptyset$ *if* φ *is a context-formula*

In the formulas we consider, for any formula $\mu\overline{X_i = \varphi_i}$ *in* ψ*, we have every occurrence of* X_i *in* ψ *guarded by a* $\langle a \rangle$*. That's why we wrote* $\{X_i\} \sqcup \mathcal{U}(\psi)$.

Such a set is well defined because the size of the formulas considered always decrease in induction calls.

2.5 Modality paths

Definition 3. *A modality path is a sequence of programs from* $\{1, 2, \bar{1}, \bar{2}\}$*. We can easily extend navigational operations to modality path. If all* $(\mathcal{T}\langle a_1\rangle \dots \langle a_n\rangle)_{1\leq i\leq n}$ *are defined then* $\mathcal{T}\langle a_1\rangle \dots \langle a_n\rangle$ *is defined and is equal to* $(\mathcal{T}\langle a_1\rangle \dots \langle a_{n-1}\rangle)\langle a_n\rangle$ *(left-associative).*

A modality path $\langle a_1\rangle \dots \langle a_n\rangle$ *is called valid on* $\mathcal{T}$ *if* $\mathcal{T}\langle a_1\rangle \dots \langle a_n\rangle$ *is defined.*

Definition 4. *A cycle in a modality path is a sub-sequence of the form* $\langle a\rangle\langle\bar{a}\rangle$*.*

Remark 3. *It has been proved in [3] that for cycle-free formulas the largest and smallest fix points collapse. Because we only consider cycle-free formulas, there is no need of a largest fix point.*

Definition 5. *We define the set* $\mathcal{P}(\varphi)$ *of modality paths of a formula* φ *as the smallest set verifying:*

- $\mathcal{P}(\varphi_1 \wedge \varphi_2) = \mathcal{P}(\varphi_1) \cup \mathcal{P}(\varphi_2)$
- $\mathcal{P}(\varphi_1 \vee \varphi_2) = \mathcal{P}(\varphi_1) \cup \mathcal{P}(\varphi_2)$
- $\mathcal{P}(\neg\varphi) = \mathcal{P}(\varphi)$
- $\mathcal{P}(\langle a\rangle \varphi) = \{S\langle a\rangle \,|S \in \mathcal{P}(\varphi)\} \cup \{\langle a\rangle\}$
- $\mathcal{P}(\mu\overline{X_i = \varphi_i} \text{ in } \psi) = \mathcal{P}(exp(\mu\overline{X_i = \varphi_i} \text{ in } \psi))$
- $\mathcal{P}(\sigma) = \{\epsilon\}$ *for* σ *a context-formula;* ϵ *represents the path of size 0.*

Definition 6. *A formula* φ *is called cycle-free if it exists* n *such that* $\forall u \in unf(\varphi), \forall p \in \mathcal{P}(u), p$ *contains less than* n *cycles.*

Lemma 1. *For every focused tree* $\mathcal{T}$ *we cannot have a valid cycle-free modality path of length greater than size*$(\mathcal{T})$*.*

Proof. Let $a_1, \dots, a_n$ be a valid cycle-free modality path, with $n \geq size(\mathcal{T})$. Because $(\mathcal{T}\langle a_1\rangle \dots \langle a_i\rangle)_i$ is valid, we can think of it as a way in the tree of $\mathcal{T}$. A way longer than the number of nodes and therefore a way that contains a cycle (in the sense of a cycle in a graph). A cycle in a tree always contains a turn back. So, $a_1, \dots, a_n$ necessarily contains a cycle (ie a pattern $\langle a\rangle\langle\bar{a}\rangle$). □

Corollary 1. *For every focused tree* $\mathcal{T}$ *and every formula* φ*, it exists a* $n_{\varphi,\mathcal{T}}$ *such that* $\forall u \in unf(\varphi), \forall p \in \mathcal{P}(u), |p| > n_{\varphi,\mathcal{T}} \Rightarrow p$ *is not valid on* $\mathcal{T}$*.*

Proof. Let φ be a cycle-free formula and $\mathcal{T}$ a focused tree. We already know there is a n such that any valid path of any unfolding of φ contains less than n cycles.

Let $\langle a_1 \rangle \dots \langle a_m \rangle$ be a valid modality path.

Let i, j be with $1 \leq i \leq j \leq n$ and $\langle a_i \rangle \dots \langle a_j \rangle$ cycle-free. Because $\langle a_1 \rangle \dots \langle a_n \rangle$ is valid we can state $\mathcal{T}' = \mathcal{T} \langle a_1 \rangle \dots \langle a_{i-1} \rangle$ and then 1 proves that $j - i \leq \mathit{size}(\mathcal{T})$. Any cycle-free subsequence is smaller in size than $\mathit{size}(\mathcal{T})$.

Now, we can cut the path at each cycle. For example
$\langle 1 \rangle \langle 2 \rangle \langle \bar{2} \rangle \langle 1 \rangle \langle 1 \rangle \langle \bar{1} \rangle \langle \bar{1} \rangle \langle \bar{1} \rangle \langle \bar{1} \rangle$ become $\langle 1 \rangle \langle 2 \rangle \mid \langle \bar{2} \rangle \langle 1 \rangle \langle 1 \rangle \mid \langle \bar{1} \rangle \langle \bar{1} \rangle \langle \bar{1} \rangle \langle \bar{1} \rangle$. There are, at most, $n + 1$ parts because there was at most n cycles. Each part is smaller than $\mathit{size}(\mathcal{T})$ so m was, at most, of size $\mathit{size}(\mathcal{T}) \times (n + 1)$. $n_{\varphi,\mathcal{T}} = \mathit{size}(\mathcal{T}) \times (n + 1)$ works. □

Remark 4. *The formulas we consider are all cycle-free.*

2.6 Focused trees

Definition 7. *The lean is the set $\mathcal{A} \cup \mathcal{B} \cup \{\langle a \rangle \varphi \mid \langle a \rangle \varphi \in \mathit{sub}(\xi) \cup \{\top\}\}$. cf [3].*

Definition 8. *We define the set $\mathcal{S}$ of tree over $\mathcal{A}$ as all finite binary trees whose labels are in $\mathcal{A}$ and attributes in $\mathcal{B}$.*

Definition 9. *A focused tree over $\mathcal{A}$ is a tree from $\mathcal{S}$ with the additional information of which node we are looking at. We write $\mathcal{F}$ for the set of all focused trees.*

For a focused tree $\mathcal{T}$ we can define the following navigational operations:

- *if $\mathcal{T}$ is centered in n and n has a father p and is a left child then $\mathcal{T} \langle \bar{1} \rangle$ is the tree of $\mathcal{T}$ but centered in p.*
- *if $\mathcal{T}$ is centered in n and n has a father p and is a right child then $\mathcal{T} \langle \bar{2} \rangle$ is the tree of $\mathcal{T}$ but centered in p.*
- *if $\mathcal{T}$ is centered in n and n has a right child p then $\mathcal{T} \langle 2 \rangle$ is the tree of $\mathcal{T}$ but centered in p.*
- *if $\mathcal{T}$ is centered in n and n has a left child p then $\mathcal{T} \langle 1 \rangle$ is the tree of $\mathcal{T}$ but centered in p.*

2.7 Types

Definition 10. *A formula is called a $\mathcal{L}ean$-formula if it can be rewritten as a formula depending on formulas in the $\mathcal{L}ean$ (in CNF form for instance?)*

Definition 11. *A set $t \subset \mathcal{L}ean$ is called a type if t does not contain both $\langle \bar{1} \rangle \top$ and $\langle \bar{2} \rangle \top$ and if it contains exactly one element from $\mathcal{A}$. A type t also needs to respect the following condition: $\langle a \rangle \varphi \in t \Rightarrow \langle a \rangle \top \in t$.*

Definition 12. *The constrained formula for a type t is $\Phi_c(t) = \bigwedge_{\varphi \in t} \varphi \wedge \bigwedge_{\varphi \in \mathcal{L}ean \setminus t} \neg\varphi$.*

Remark 5. *Given a type t and a $\mathcal{L}ean$-formula φ we have either $\Phi_c(t) \Rightarrow \varphi$ or $\Phi_c(t) \Rightarrow \neg\varphi$.*

Definition 13. *Two types x and y are compatible for $a \in \{1, 2, \bar{1}, \bar{2}\}$ if, for every $\langle a \rangle \varphi \in \mathcal{L}ean$, we have $\langle a \rangle \varphi \in x \Leftrightarrow (\Phi_c(y) \Rightarrow \varphi)$. For $a \in \{1, 2\}$, we have $\Delta_a(x, y)$ iff x is compatible with y for a and y with x for $\bar{a}$*

3 An Automata Perspective

3.1 Automata over trees with attributes

Because trees with labels from $\mathcal{A}$ and attributes from $\mathcal{B}$, can be seen like trees with labels in $\mathcal{A} \times 2^{\mathcal{B}}$; we see automata for trees with attributes like automata over the alphabet $\mathcal{A} \times 2^{\mathcal{B}}$. It might also be useful to erase some attributes by transforming all rules labels from $a, x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{|\mathcal{B}|}$ to $a, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{|\mathcal{B}|}$ and see the automaton for an alphabet $\mathcal{A} \times 2^{\mathcal{B} \setminus \{x_i\}}$.

3.2 Automaton for a formula

We build the automaton $(Q, \mathcal{A}, \delta, \mathcal{S})$ for the formula ξ with:

- Q the set of types
- For any tuple of types x, y, z, any label $c \in \mathcal{A}$ and any set of attributes $\mathcal{O} \subset \mathcal{B}$ we have:
  - If $\Phi_c(x) \Rightarrow \langle 1 \rangle \top \wedge \langle 2 \rangle \top$ then δ contains $$\begin{tikzcd}[column sep=small] & c \arrow[dl, dash] \arrow[dr, dash] & \\ y & & z \end{tikzcd}$$ $\rightarrow x$ iff $\Delta_1(x, y) \wedge \Delta_2(x, z) \wedge \mathcal{L}(x) = c \wedge \mathcal{O} = \mathcal{S}(x)$
  - If $\Phi_c(x) \Rightarrow \neg \langle 1 \rangle \top \wedge \langle 2 \rangle \top$ then δ contains $$\begin{tikzcd}[column sep=small] & c \arrow[dl, dash] \arrow[dr, dash] & \\ y & & {} \end{tikzcd}$$ $\rightarrow x$ iff $\Delta_1(x, y) \wedge \mathcal{L}(x) = c \wedge \mathcal{O} = \mathcal{S}(x)$

  - If $\Phi_c(x) \Rightarrow \langle 1 \rangle \top \wedge \neg \langle 2 \rangle \top$ then δ contains $\overset{c}{\diagup\diagdown}_{z} \to x$ iff $\Delta_2(x,z) \wedge \mathcal{L}(x) = c \wedge \mathcal{O} = \mathcal{S}(x)$
  - If $\Phi_c(x) \Rightarrow \neg \langle 1 \rangle \top \wedge \neg \langle 2 \rangle \top$ then δ contains $\overset{c}{\diagup\diagdown} \to x$ iff $\mathcal{L}(x) = c \wedge \mathcal{O} = \mathcal{S}(x)$

- $Q_f = \{q \in Q \;/\; \Phi_c(q) \Rightarrow \mu X = \xi \vee \langle 1 \rangle X \vee \langle 2 \rangle X \text{ in } X \wedge \neg \langle \bar{1} \rangle \top \wedge \neg \langle \bar{2} \rangle \top\}$.

It's easy to build an algorithm that, given a formula ξ, builds the above automaton in single exponential time.

3.3 Consequences

The translation to NDFTA can be done in single exponential time.

4 Proof

4.1 Existence of run

Lemma 2. *For any tree T from $[\![\xi]\!]$, it exists a labeling of the nodes of T with states of Q that is consistent with the transitions built for ξ.*

Proof. Let T be a tree. We introduce a labeling and then we show that it is consistent.

The labeling $t \to q_t$ is defined like this : for t a node and for $\varphi \in \mathcal{L}ean$, $\varphi \in q_t$ iff $t \in [\![\varphi]\!]$.

There are four cases to consider but they are redundant. So, we will not consider the cases of trees with one child.

In the case of a node d with no children. The focused tree $\mathcal{T}$ focused on d is in $[\![\Phi_c(q_d)]\!]$. So, d respects $\mathcal{S}(q_d)$ and $\mathcal{L}(q_d)$ and the automaton can jump to q_d.

In the case of a node d, where d has a left child f_1 and a right f_2. We will show that if the automaton can jump to q_{f_a} when it reads f_a for $a \in \{1,2\}$ then it can jump to q_d when it reads d. As before, d respects $\mathcal{S}(q_d)$ and $\mathcal{L}(q_d)$. We need also to show that $\Delta_a(q_d, q_{f_a})$ for $a \in \{1,2\}$. By definition:

$$\Delta_a(q_d, q_{f_a}) \;\Leftrightarrow\; \begin{cases} \langle a \rangle \varphi \in q_d & \Leftrightarrow \quad \Phi_c(q_{f_a}) \Rightarrow \varphi \quad \text{for } \langle a \rangle \varphi \in \mathcal{L}ean \\ \langle \bar{a} \rangle \varphi \in q_{f_a} & \Leftrightarrow \quad \Phi_c(q_d) \Rightarrow \varphi \quad \text{for } \langle \bar{a} \rangle \varphi \in \mathcal{L}ean \end{cases}$$

and for $\langle a \rangle \varphi \in \mathcal{L}ean$ as we already have $f_a \in [\![\Phi_c(q_{f_a})]\!]$ so

$$\langle a \rangle \varphi \in q_d \Leftrightarrow d \in [\![\langle a \rangle \varphi]\!] \Leftrightarrow f_a \in [\![\varphi]\!] \Leftrightarrow f_a \in [\![\varphi \wedge \Phi_c(q_{f_a})]\!]$$

$$\langle a \rangle \varphi \in q_d \Leftrightarrow [\![\varphi \wedge \Phi_c(q_{f_a})]\!] \neq \emptyset \Leftrightarrow \neg(\Phi_c(q_{f_a}) \Rightarrow \neg\varphi) \Leftrightarrow \Phi_c(q_{f_a}) \Rightarrow \varphi$$

At the same time for $\langle \bar{a} \rangle \varphi \in \mathcal{L}ean$ we already have $d \in [\![\Phi_c(q_d)]\!]$ so

$$\langle \bar{a} \rangle \varphi \in q_{f_a} \Leftrightarrow f_a \in [\![\langle \bar{a} \rangle \varphi]\!] \Leftrightarrow d \in [\![\varphi]\!] \Leftrightarrow d \in [\![\varphi \wedge \Phi_c(q_d)]\!]$$

$$\langle \bar{a} \rangle \varphi \in q_{f_a} \Leftrightarrow [\![\varphi \wedge \Phi_c(q_d)]\!] \neq \emptyset \Leftrightarrow \neg(\Phi_c(q_d) \Rightarrow \neg\varphi) \Leftrightarrow \Phi_c(q_d) \Rightarrow \varphi$$

As expected, we got:

$$\Delta_1(q_d, q_{f_1}) \wedge \Delta_2(q_d, q_{f_2}) = \top$$

□

4.2 The verification function $\mathcal{V}$

Now, we consider φ, a $\mathcal{L}ean$-formula, and $t \to q_t$ a run.

As we want to prove that states define the correct truth assignment, we build a function $\mathcal{V}$ to check, truth assignments by passing through, at most, k modalities.

Definition 14. *We define $\mathcal{V}$ like this :*

- *If $k = 0$ then* $\mathcal{V}\left(\varphi, \begin{tikzcd} & d \arrow[d, no head] & \\ & t \arrow[dl, no head] \arrow[dr, no head] & \\ a & & b \end{tikzcd}, 0\right) = (\Phi_c(q_t) \Rightarrow \varphi)$

- *If $\varphi = \top$ then* $\mathcal{V}(\varphi, \bullet, \bullet) = \varphi = \top$

- *If $\varphi \in \mathcal{A} \cup \mathcal{B}$ formula then* $\mathcal{V}(\varphi, \mathcal{T}, \bullet) = \varphi \in q_t$

- $\mathcal{V}(\varphi_1 \vee \varphi_2, \mathcal{T}, k+1) = \mathcal{V}(\varphi_1, \mathcal{T}, k+1) \vee \mathcal{V}(\varphi_2, \mathcal{T}, k+1)$

- $\mathcal{V}(\varphi_1 \wedge \varphi_2, \mathcal{T}, k+1) = \mathcal{V}(\varphi_1, \mathcal{T}, k+1) \wedge \mathcal{V}(\varphi_2, \mathcal{T}, k+1)$

- $\mathcal{V}(\neg\varphi, \mathcal{T}, k+1) = \neg\mathcal{V}(\varphi, \mathcal{T}, k+1)$

- $\mathcal{V}(\langle a \rangle \varphi, \mathcal{T}, k+1) = \begin{cases} \mathcal{V}(\varphi, \mathcal{T}\langle a \rangle, k) & \textit{if } \mathcal{T}\langle a \rangle \in q_t \\ \bot & \textit{if } \neg\mathcal{T}\langle a \rangle \in q_t \end{cases}$

- $\begin{array}{rcl} \mathcal{V}(\mu\overline{X_i = \varphi_i} \textit{ in } \psi, \mathcal{T}, k+1) & = & \mathcal{V}(exp(\mu\overline{X_i = \varphi_i} \textit{ in } \varphi_i), \mathcal{T}, k+1) \\ & = & \mathcal{V}(\varphi_{[X_i/\mu\overline{X_i=\varphi_i} \textit{ in } \varphi_i]}, \mathcal{T}, k+1) \end{array}$.

Remark 6. *$\mathcal{V}$ is defined by induction on k, next on the set of modality-free variables, next on the size of the formula.*

Because $\mathcal{V}$ is defined recursively we can make proofs by induction on it.

Lemma 3. *At φ and $\mathcal{T}$ constant, the function $k \to \mathcal{V}(\varphi, \mathcal{T}, k)$ is constant.*

Proof. We prove that $\mathcal{V}(\varphi, \mathcal{T}, k) = \mathcal{V}(\varphi, \mathcal{T}, 0)$ by induction on k, next on the set of modality-free variables and finally on the size of the formulas.

The base case $k = 0$ is immediate.

Consider k, φ, $\mathcal{T} =$

$$\begin{tikzcd}
& d \arrow[d, dash] & \\
& t \arrow[dl, dash] \arrow[dr, dash] & \\
a & & b
\end{tikzcd}$$

- if φ is a context formula then $\mathcal{V}$ does not depend on k.
- if $\varphi = \varphi_1 \wedge \varphi_2$, $\varphi = \varphi_1 \vee \varphi_2$ or $\varphi = \neg\varphi'$. Only the size of formulas changes (and decreases), the result holds by induction.
- if $\varphi = \langle a \rangle \psi$, we have

  $$\mathcal{V}(\langle a \rangle \psi, \mathcal{T}, k+1) = \mathcal{V}(\psi, \mathcal{T}\langle a \rangle, k) = \mathcal{V}(\psi, \mathcal{T}\langle a \rangle, 0) = \mathcal{V}(\langle a \rangle \psi, \mathcal{T}, 1)$$

  but by definition of Δ_a we have

  $$\mathcal{V}(\psi, \mathcal{T}\langle a \rangle, 0) \Leftrightarrow \langle a \rangle \psi \in q_t$$

  and by definition of $\mathcal{V}$ we have

  $$\langle a \rangle \psi \in q_t \Leftrightarrow \mathcal{V}(\langle a \rangle \psi, \mathcal{T}, 0)$$

  Finally we have

  $$\mathcal{V}(\langle a \rangle \psi, \mathcal{T}, k+1) = \mathcal{V}(\langle a \rangle \psi, \mathcal{T}, 0)$$

- if $\varphi = \mu\overline{X_i = \varphi_i}$ *in* ξ then X_i cannot appear with modality in ξ so the set of modality-free variables decrease and we have the result by induction.

$\square$

Remark 7. *As k does not play a role in the definition of $\mathcal{V}$ we can refer to $\mathcal{V}(\varphi, \mathcal{T}, k)$ as $\mathcal{V}(\varphi, \mathcal{T})$.*

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

Lemma 4. *For every focused tree and every $\mathcal{L}$ean-formula φ we have $\mathcal{V}(\varphi, \mathcal{T}) \Leftrightarrow \mathcal{T} \in [\![\varphi]\!]$.*

Proof. Let k be such that $k > n(\varphi, \mathcal{T})$. That means it does not exist any valid modality paths p from $\mathcal{P}(u)$ where u is an unfolding of φ and p is of size k.

We now show that $\mathcal{V}(\varphi, \mathcal{T}) = \mathcal{V}(\varphi, \mathcal{T}, k) = (\mathcal{T} \in [\![\varphi]\!])$ for $k \geq n(\varphi, \mathcal{T})$ by induction on the order used to define $\mathcal{V}$.

- if φ is a context formula then clearly $\mathcal{V}(\varphi, \mathcal{T}, k) = \mathcal{T} \in [\![\varphi]\!]$.
- if $\varphi = \varphi_1 \wedge \varphi_2$, because $\mathcal{V}(\varphi, \mathcal{T}, k) = \mathcal{V}(\varphi_1, \mathcal{T}, k) \wedge \mathcal{V}(\varphi_2, \mathcal{T}, k)$ and $[\![\varphi_1 \wedge \varphi_2]\!] = [\![\varphi_1]\!] \cap [\![\varphi_2]\!]$ we have by induction $\mathcal{V}(\varphi, \mathcal{T}, k) = (\mathcal{T} \in [\![\varphi]\!])$. We have the induction property for k, because any path of any unfolding of φ_1 or φ_2 is a path for an unfolding of φ.
- $\varphi = \varphi_1 \vee \varphi_2$ or $\varphi = \neg\varphi'$ it is the same.
- if $\varphi = \langle a \rangle \psi$. If $\mathcal{T} \langle a \rangle$ then we have $\mathcal{V}(\varphi, \mathcal{T}, k+1) = \mathcal{V}(\psi, \mathcal{T} \langle a \rangle, k)$ and $\mathcal{V}(\psi, \langle a \rangle \mathcal{T}, k) = \mathcal{T} \langle a \rangle \in [\![\psi]\!] = \mathcal{T} \langle a \rangle \langle \bar{a} \rangle \in [\![\varphi]\!] = \mathcal{T} \in [\![\varphi]\!]$. If $\neg\mathcal{T} \langle a \rangle$ then $\mathcal{V}(\varphi, \mathcal{T}, k+1) = \bot$ and $\mathcal{T} \notin [\![\langle a \rangle \varphi]\!] = \{\mathcal{T} \langle \bar{a} \rangle / \mathcal{T} \in [\![\varphi]\!]\}$ so we always have $\mathcal{V}(\varphi, \mathcal{T}, k+1) = \mathcal{T} \in [\![\varphi]\!]$. We have the induction property for k, because any path of any unfolding of ψ of size k is a path for an unfolding of φ of size $k+1$.
- if $\varphi = \mu\overline{X_i = \varphi_i}\ in\ \psi$ then $\mathcal{V}(\varphi, \mathcal{T}, k) = \mathcal{V}(exp(\mu\overline{X_i = \varphi_i}\ in\ \psi), \mathcal{T}, k) = \mathcal{T} \in [\![exp(\mu\overline{X_i = \varphi_i}\ in\ \psi)]\!] = \mathcal{T} \in [\![\varphi]\!]$. We have the induction property for k, because any path of any unfolding of $exp(\mu\overline{X_i = \varphi_i}\ in\ \psi)$ is a path for an unfolding of φ.

□

Lemma 5. *A tree is accepted by the automaton made for the formula ψ iff this tree respect ψ.*

Proof. Given a tree $\mathcal{T}$, it exists a run on $\mathcal{T}$, by definition of Q_f the run is accepted iff we have $\mathcal{V}(\mu X = \xi \vee \langle 1 \rangle X \vee \langle 2 \rangle X \text{ in } X \wedge \neg \langle \bar{1} \rangle \top \wedge \neg \langle \bar{2} \rangle \top, \mathcal{T}, 0)$ iff $\mathcal{T} \in [\![\mu X = \xi \vee \langle 1 \rangle X \vee \langle 2 \rangle X \text{ in } X \wedge \neg \langle \bar{1} \rangle \top \wedge \neg \langle \bar{2} \rangle \top]\!]$. □

5 Containment of Path Formulas

Attributes from $\mathcal{B}$ are needed for translating XPath formulas with XML attributes. But, in the context of XPath, they can also be useful to describe where the context of the evaluation started and stating intersection, containment, union of path formulas like:

$$\texttt{self::a/descendant::b} \cap \texttt{self::b/descendant::b} = \emptyset. \qquad (1)$$

5.1 Query Automata

The idea of using query automata to recognize the sets of solution for CXpath formulas was already in [2]. It is easy to adapt the above construction to build a query automaton. For instance, we could state that the set of accepting states that would be $\{s \in Q | \Phi_c(s) \Rightarrow \xi\}$. Here we will introduce a more powerful tool: the binary relation automaton.

5.2 Binary relation over binary tree with automata

Definition 15. *Let $\mathcal{L}$ be a regular tree language over an alphabet $\mathcal{A} \times \{0,1\}^2$, T a binary tree, l a function labeling T with the alphabet $\mathcal{A}$.*

If a is a function labeling T with the alphabet $\{0,1\}^2$, we say that a is an annotation. For an annotation we define a function labeling T with $\mathcal{A} \times \{0,1\}^2$ by $l_a(n) = (l(n), a(n))$

For any annotation a, such that T labeled by l_a is in $\mathcal{L}$, we ensure that it exists x and y with either $y = x \wedge a(x) = (1,1)$ and $z \neq x \Rightarrow a(z) = (0,0)$ or with $a(x) = (1,0)$, $a(y) = (0,1)$ and $\forall z, z \neq y \wedge z \neq x \Rightarrow a(z) = (0,0)$. Given a x and a y there is only one annotation satisfying those requirements so we can write $a_{(x,y)}$.

Because checking that an annotation is of the form $a_{(x,y)}$ - ie the annotation that has on each coordinate one node with a 1 - can be done by a regular tree language, we can suppose there are only annotation a of this form that produces l_a in $\mathcal{L}$.

$\mathcal{R}_T$ is the binary relation over the nodes of T defined like this by $x\mathcal{R}_T y$ iff T labeled by $l_{a_{(x,y)}}$ is in $\mathcal{L}$.

Proposition 1. *Given a tree T,and a labeling l, suppose we have $\mathcal{L}_1$ defining a relation $\mathcal{R}_{T,1}$ and $\mathcal{L}_2$ defining $\mathcal{R}_{T,2}$ then $\mathcal{L}_1 \cap \mathcal{L}_2$ define $x\mathcal{R}_T y \Leftrightarrow x\mathcal{R}_{T,1} y \wedge x\mathcal{R}_{T,2} y$.*

Proof. Let x and y be two nodes. We have $x\mathcal{R}y$ iff T labeled with $l_{a_{(x,y)}}$ is in $\mathcal{L}_1 \cap \mathcal{L}_2$ iff T labeled with $l_{a_{(x,y)}}$ is in $\mathcal{L}_1$ and T labeled with $l_{a_{(x,y)}}$ is in $\mathcal{L}_2$ iff $x\mathcal{R}_1 y$ and $x\mathcal{R}_2 y$. □

Proposition 2. *Given a tree T,and a labeling l, suppose we have $\mathcal{L}_1$ defining a relation $\mathcal{R}_{T,1}$ and $\mathcal{L}_2$ defining $\mathcal{R}_{T,2}$ then $\mathcal{L}_1 \cup \mathcal{L}_2$ define $x\mathcal{R}_T y \Leftrightarrow x\mathcal{R}_{T,1} y \vee x\mathcal{R}_{T,2} y$.*

Proposition 3. *Given a tree T,and a labeling l, suppose we have $\mathcal{L}$ defining a relation $\mathcal{R}$ then $\bar{\mathcal{L}}$ define $\neg x\mathcal{R}_T y$.*

5.3 Nominals

To make use of binary relation with automata it might be useful to have attributes for which we are sure they are only present once in the tree. The most easiest way is to add to the formula. To ensure that the attribute c is present only one in the formula, we write :

$$\neg\mu X = \langle 1 \rangle\,(X \vee c) \vee \langle 2 \rangle\,(X \vee c)\ in\ X \wedge \neg\mu X = \langle \bar{1} \rangle\,(X \vee c) \vee \langle \bar{2} \rangle\,(X \vee c)\ in\ X \wedge c$$

An attributes for which we made sure it was unique can be called a nominal.

5.4 Methodology

We consider $\varphi = context \wedge \langle 1 \rangle\,(\mu X = (a \wedge \langle 2 \rangle\,(select \wedge b)) \vee \langle 2 \rangle\, X\ in\ X)$ where a and b are labels and *context* and *select* are attributes. φ selects all trees where a node with the attribute *context* has a child a who has a brother b with an attribute *select*. If we ensure *context* and *select* are nominals then this correspond to the XPath query `child::a/following-sibling::b`. The test the containment of this query into $\psi = context \wedge \langle 1 \rangle\,(\mu X = (a \wedge select) \vee \langle 2 \rangle\, X\ \ in\ X) \vee \langle 1 \rangle\,(\mu X = b \vee \langle 2 \rangle\, X\ in\ X)$ corresponding to `self::*[child::b]/child::a` can be done by building the automata for φ and $\neg\psi$, make the intersection of the two and then test the emptiness. The automaton will also permit to give counter-example.

5.5 Translation of XPath to formula with context and select nominals

In order to define XPath into path automata we need to add the attributes context and select. Context was already introduced in [3] but it was not a nominal. Once the formula is translated with a context into φ then to add select we just use the formula $\varphi \wedge @select$. Then we make $@context$ and $@select$ nominals and plunge the formula.

6 Practical Algorithm

The practical algorithm does not build the exact automaton describe above but an equivalent one. A state is not a type but a set of type, this leads to memory efficiency and speed improvement because many types share the same kind of children. We often perform determinization of the automata we build because deterministic automata have a minimal form that is, in general very compact. (In the tests, all minimal deterministic representation of automata were smaller than the automata that would have been built with the above description, despite a theoretical exponential blow-up).

Algorithm 1 BUILD

```
Input: $S$ a set of types
Output: The name of the state created
1: Memoize $S$
2: $name \leftarrow newname()$
3: for all $t \in S$ do
4:     $left \leftarrow BUILD(SUCCS(j, left))$
5:     $right \leftarrow BUILD(SUCCS(k, right))$
6:     $rules \leftarrow rules \cup \{ \overset{\mathcal{L}(i)}{left \quad right} \rightarrow name\}$
7: end for
8: return $name$
```

7 Results and Conclusion

The implementation of tree automata was first a naive translation of their pseudo-code in Java. Because it was too slow to be useful some parts of the code were rewritten to improve speed. As explained in [5], a much better representation can be chosen for the automata, leading to great speed improvements and more memory efficiency. But, even without a complex representation of automata, rewriting the automata management with a precise control over the memory used and a simple profiling of the code can lead to, at least, a gain of a factor 5 in speed and in memory footprint. Because

Algorithm 2 SUCCS

```
Input: $t$ a type, $side \in \{left, right\}$
Output: $S$ the set of types compatible with $t$ on the side $side$
 1: $repr = t \cap USEFULLFOR(side)$
 2: Memoize $(repr, side)$
 3: $res = Fixpoint$
 4: for all $\varphi \in \mathcal{L}ean$ do
 5:     if $\varphi = \langle side \rangle\, \psi \in t$ or $(\psi = \langle \bar{side} \rangle \varphi$ and $\varphi \in t)$ then
 6:         $res \leftarrow \{e \in res, \psi \in e\}$
 7:     else
 8:         $res \leftarrow \{e \in res, \psi \notin e\}$
 9:     end if
10: end for
11: return $BUILD(res)$
```

the other part of the code is very well optimized, in small test cases the old algorithm is faster, but, in large test cases the algorithm using automata out-powers the old one.

```
select("*[not (ancestor::*/descendant::b[ancestor::i])]")
select("*[not (ancestor::*/descendant::img[not (ancestor::body)])]")
select("*[not (ancestor::*/descendant::img[not (parent::p)])]")
select("*[not (ancestor::*/descendant::p[parent::a])]")
select("*[not (ancestor::*/descendant::b[ancestor::i])]")
select("*[not (ancestor::*/descendant::img[parent::p])]")
select("*[not (ancestor::*/descendant::img[*])]")
select("*[not (ancestor::*/descendant::a[ancestor::a])]")
select("*[not (ancestor::*/descendant::div[parent::b])]")
select("*[not (ancestor::*/descendant::h1[ancestor::h2])]")
```

Figure 1: Sample Formulas.

For these reasons a complete comparison of the two algorithm is not necessary nor useful. But, for example, with the xhtml DTD, applying all the XPath expressions of Figure 1 is done by the algorithm in 30s whereas the old XML logic solver was suspended after several hours of calculation.

References

[1] Initial proposal submission of the ANR project TYPEX, ANR-11-BS02-007., 2012.

[2] Nadime Francis, Claire David, and Leonid Libkin. A Direct Translation from XPath to Nondeterministic Automata. In *5th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2011.

[3] Pierre Genevès, Nabil Layaïda, and Alan Schmitt. Efficient static analysis of XML paths and types. *SIGPLAN Not.*, 42:342–351, June 2007.

[4] Nils Gesbert, Pierre Genevès, and Nabil Layaïda. A Logical Approach To Deciding Semantic Subtyping - Supporting function, intersection, negation, and polymorphic types. July 2013.

[5] Hendrik Maryns and Universität Tübingen. On the implementation of tree automata: Limitations of the naive approach. In *In Proc. 5th Int. Treebanks and Linguistic Theories Conference (TLT 2006*, pages 235–246, 2006.