Composing Monadic Queries in Trees

Emmanuel Filiot
INRIA Futurs, Lille

Joachim Niehren
INRIA Futurs, Lille

Jean-Marc Talbot
LIFL, Lille

Sophie Tison
LIFL, Lille

Abstract

Node selection in trees is a fundamental operation to XML databases, programming languages, and information extraction. We propose a new class of querying languages to define *n*-ary node selection queries as compositions of monadic queries. The choice of the underlying monadic querying language is parametric. We show that compositions of monadic MSO-definable queries capture *n*-ary MSO-definable queries, and distinguish an MSO-complete *n*-ary query language that enjoys an efficient query answering algorithm.

1 Introduction

Node selection in trees [12] is a fundamental operation to XML databases, programming languages, and information extraction. Node selecting captures the matching aspect of tree transformations. Iterated node selection can be used to navigate through input trees while producing output data structures.

From the *database perspective*, node selecting is usually viewed as a querying problem [13, 15, 10]. The W3C standard querying language XPath provides descriptions of *monadic queries*, i.e. queries that define sets of nodes in trees. XPath queries are used by the W3C standard languages XQuery and XSLT for defining XML document transformations.

Modern *programming languages* support node selection in trees via *pattern matching*, for instance all functional programming languages of the ML family (Caml, SML, Haskell). Tree pattern with *n* capture variables define *n-ary queries*, i.e. queries that select sets of *n*-tuples of nodes. The XML programming languages XDuce [11] and CDuce [4, 2] support more expressive *recursive tree pattern*.

Information extraction tasks for the Web can frequently be reduced to defining *n*-ary queries in HTML or XML trees. Gottlob et. al. [9] proposed monadic Datalog as querying language for this purpose, and show that it captures monadic MSO-definable queries [8]. Their Lixto system [6, 1] provides a visual interface by which to specify monadic queries in monadic Datalog, and to compose them into *n*-ary queries. Composed monadic queries are defined in Elog, a binary Datalog language.

We propose a new class of querying languages to define *n*-ary node selection queries as compositions of monadic queries. The choice of the underlying monadic querying language is parametric. We show that compositions of monadic MSO-definable queries capture *n*-ary MSO-definable queries, and distinguish an MSO-complete *n*-ary query language that enjoys efficient query answering algorithms. Moreover, our language allow to compose different monadic query languages, for instance a monadic query defined by a Datalog program with another monadic query defined by an XPath node formula.

Compositions of monadic MSO-definable queries are relevant to information extraction. They might be useful to approach an open question in the context of the Lixto system [6], of how to enhance such system by machine learning techniques. Given a composition formula, and examples for n-tuples that are to be selected, one can use existing learning algorithms for monadic MSO-definable queries [3], in order to infer n-ary MSO definable queries.

The paper is organized as follows. We recall the definitions of *n*-ary MSO definable queries in tree in *Section 2*, introduce languages of compositions of monadic queries in *Section 3*, and discuss some instances in *Section 4*. We study their expressiveness in *Section 5* and their algorithmic complexity in *Section 6*, including algorithms for the model-checking and the query answering problems, and prove the satisfiability problem to be NP-hard. Finally in *Section 7* we propose a fragment of it, study its expressiveness and give an efficient algorithm for the query answering problem.

2 Node Selection Queries in Trees

We recall the definition of *n*-ary MSO-definable queries in trees. We develop our theory for binary trees. This will be sufficient to deal with unranked trees, since unranked trees can be viewed as binary trees via a `firstchild` – `nextsibling` encoding [12].

We consider binary trees as acyclic digraphs with labeled nodes and ordered children. We start with a finite set Σ of node labels. A binary Σ-tree $t \in T_\Sigma$ is a finite rooted acyclic directed graph, whose nodes are labeled in Σ. Every node is connected to the root by a unique path. All nodes of binary trees either have 0 or 2 children. Nodes without children are called *leaves*. All other nodes have a distinguished first and second child.

We write $\mathtt{root}(t)$ for the root of tree t and $\mathtt{nodes}(t)$ for the set of nodes of tree t and $\mathtt{edges}(t) \subseteq \mathtt{nodes}(t)^2$ for the set of edges of t. For all labels $a \in \Sigma$, we write $\mathtt{lab}_a(t) \subseteq \mathtt{nodes}(t)$ for the subset of nodes of t labeled by a. Given two nodes $v_1, v_2 \in \mathtt{nodes}(t)$ we call v_2 a *child* of v_1 and write $v_1 \lhd v_2$ iff there exists an edge from v_1 to v_2, i.e., if $(v_1, v_2) \in \mathtt{edges}(t)$.

The *descendant* relation $\lhd^*$ on nodes is the reflexive transitive closure of the child relation $\lhd$.

The *subtree of tree t rooted by node $v \in \mathtt{nodes}(t)$* is the tree denoted by $t|_v$ that satisfies:

$$\begin{array}{rcll} \mathtt{nodes}(t|_v) & = & \{v' \in \mathtt{nodes}(t) \mid v \lhd^* v'\} & \\ \mathtt{edges}(t|_v) & = & \mathtt{edges}(t) \cap \mathtt{nodes}(t|_v)^2 & \\ \mathtt{root}(t|_v) & = & v & \\ \mathtt{lab}_a(t|_v) & = & \mathtt{lab}_a(t) \cap \mathtt{nodes}(t|_v) & \forall a \in \Sigma \end{array}$$

Definition 1. *Let $n \in \mathbb{N}$. An n-ary query in binary trees over Σ is a function q that maps trees $t \in T_\Sigma$ to set of n-tuples of nodes, such that $\forall t \in T_\Sigma : q(t) \subseteq \mathtt{nodes}(t)^n$. Moreover, we require q to be closed under tree-isomorphism, i.e. $h(q(t)) = q(h(t))$ for a tree isomorphism h.*

Simple examples for monadic queries in binary trees over Σ are the functions $\mathtt{lab}_a$ that map trees t to the sets of nodes of t that are labeled by a for $a \in \Sigma$. The binary query $\mathtt{descendant}$ relates nodes v to their descendants, i.e. $\mathtt{descendant}(t) = \{(v, v') \in \mathtt{nodes}(t)^2 \mid v \lhd^* v'\}$.

Definition 2. *A* query language *L over alphabet Σ is a pair $L = (\mathtt{N}, [\![.]\!])$ where $\mathtt{N}$ is a set of names and $[\![.]\!]$ an interpretation function mapping names $c \in \mathtt{N}$ to queries $[\![c]\!]$ in Σ-trees.*

The monadic second-order logic (MSO) in trees is a query language that is widely accepted as the yardstick for comparing the expressiveness of XML-query languages [9, 15]. This is because of the close correspondence between MSO, tree automata, and regular tree languages [17]. Every MSO formula with n free node variables defines an n-ary query.

In MSO, binary trees $t \in T_\Sigma$ are seen as *logical structures* with domain $\mathtt{nodes}(t)$. The signature $\mathbb{T}$ of this structure contains symbols for the binary relations $\mathtt{child}_1$ and $\mathtt{child}_2$ and the unary relations $\mathtt{lab}_a$ for all $a \in \Sigma$.

Let x, y, z range over a countable set of first-order variables and X over a countable set of monadic second-order variables. Formulas ϕ of MSO have the following abstract syntax, where $a \in \Sigma$:

$$\begin{array}{rcl} \phi & ::= & p(x) \mid \mathtt{child}_1(x,y) \mid \mathtt{child}_2(x,y) \\ & & \mid \mathtt{lab}_a(x) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \forall x \phi \mid \forall X \phi \end{array}$$

A variable assignment α into a tree t maps first-order variables $\mathtt{nodes}(t)$ and second-order variables to subsets of $\mathtt{nodes}(t)$. We define the validity of formulas ϕ in trees t under variable assignments α in the usual Tarskian manner, and write $t, \alpha \models \phi$ in this case. The *first-order logic* FO is obtained from MSO by omitting the set quantification. Actually the notations FO and MSO stands for FO[$\mathbb{T}$] and MSO[$\mathbb{T}$] respectively, i.e. formulae over the vocabulary $\mathbb{T}$.

We view MSO as a query language. The names of n-ary queries are MSO formulas $\phi(x_1, \ldots, x_n)$ with n free first-order variables $x_1, \ldots, x_n$. These define the following queries:

$$[\![\phi(x_1, \ldots, x_n)]\!](t) = \{(\alpha(x_1), \ldots, \alpha(x_n)) \mid t, \alpha \models \phi\}$$

Definition 3. *An n-ary query is* MSO definable *if it is equal to some query $[\![\phi(x_1, \ldots, x_n)]\!]$.*

Unfortunately [5] shows that the satisfiability problem is not fixed-parameter tractable, i.e. there exists no polynomial p and elementary function f such that we can decide in time $O(f(|\phi|)\ p(|t|))$ whether an monadic MSO-formula ϕ is satisfiable in the tree t. However, there exists query languages that can express all MSO-definable queries, which have polynomial-time combined complexity: e.g. monadic queries defined by successful runs of tree automata have exactly the power of MSO in defining monadic queries but deciding the non-emptyness of a monadic query is in polynomial-time w.r.t. combined complexity [14].

Let us define some algorithmic tasks for query languages $(\mathtt{N}, [\![.]\!])$ that are common to database theory:

- *model-checking*: given a query name c, a tree t and an n-tuple $(v_1, \ldots, v_k) \in \mathtt{nodes}(t)^k$, does $(v_1, \ldots, v_k) \in [\![c]\!](t)$ hold ?
- *query answering*: given a query name c and a tree t, return $[\![c]\!](t)$. An expected complexity might be polynomial in the number of solutions.
- *satisfiability (over a fixed tree)*: given a query name c and a tree t, does $[\![c]\!](t) \neq \emptyset$ hold ?

Unranked trees are like binary trees, except that all nodes may have arbitrarily many ordered children. The next-sibling of a node is the successor of the same parent in the sibling ordering.

Unranked trees can be encoded as binary trees by only using edges for the first-child and next-sibling relations. Fig. 1 gives a DTD, an unranked tree matching this DTD and its first-child next-sibling encoding t. A simple binary query on that tree is to select all pairs of name and title of the same book. It can be expressed with respect to the binary encoding by the following MSO formula with two free variables y, z:

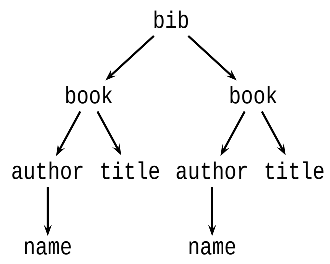
$$\exists x\ (\mathtt{lab}_{\mathtt{author}}(x) \wedge \mathtt{child}_1(x,y) \wedge \mathtt{child}_2(x,z))$$

3 Composing Monadic Queries

Query languages for monadic queries in trees have been widely studied by the database community in the last few years. See [12] for a comprehensive overview. Languages for n-ary queries are less frequent but have started to arise with the XML programming languages

DTD	unranked tree	`fc-ns` encoding

```
<!ELEMENT bib (book)*>
<!ELEMENT book (author,title)>
<!ELEMENT author (name)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```



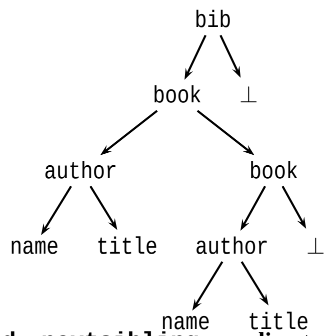


Figure 1. A DTD, an unranked tree matching the DTD and its `firstchild` − `nextsibling` encoding t.

XDuce and CDuce [11, 2, 4] as well as with information extraction tools such as Lixto [8, 6].

In this paper, we propose a new class of languages for defining n-ary queries by composition of monadic queries. We leave the choice of the underlying monadic querying language parametric, so the reader may choose his prefered monadic querying language, and extend it to an n-ary query language by query composition. The composition operator is motivated by Lixto's way of defining n-ary queries [6].

The principle of composition is quite simple: a composition of two monadic queries first selects a node answering the first sub-query and then launches the second sub-query at that node. All nodes seen meanwhile can be memoized and returned in an output tuple.

We start from a language L of monadic queries c and an infinite set $x,y,z \in \mathtt{Var}$ of variables. We then define compositions of monadic queries on basis of the composition operator that we write as the dot '.' . Informally a composition query $c_1(x_1).c_2(x_2)$ on a tree t will first bind x_1 nondeterministically to some node $v_1 \in [\![c_1]\!](t)$, and then launch query c_2 in the subtree $t|_{v_1}$ rooted at v_1 in order to bind x_2 to some node $v_2 \in [\![c_2]\!](t|_{v_1})$.

For expressivity reasons – that is to capture MSO as soon as the monadic query language capture MSO – we add conjunction, disjunction and projection to our composition language [1].

Given a monadic query language $L = (\mathtt{N}, [\![.]\!])$, *composition formulae* $\phi \in \mathcal{C}(L)$ are defined by the following abstract syntax:

$$
\begin{array}{lcll}
\phi & ::= & & \textit{composition formula} \\
 & & \top & \\
 & | & c(x).\phi & \textit{composition, } c \in \mathtt{N}, x \in \mathtt{Var} \\
 & | & \phi \wedge \phi & \textit{conjunction} \\
 & | & \phi \vee \phi & \textit{disjunction} \\
 & | & \exists x\ \phi & \textit{projection}
\end{array}
$$

Given a composition formula ϕ, we denotes by $\mathrm{FV}(\phi)$ the set of free variables of ϕ. We will often write $c(x)$ instead of $c(x).\top$. The set of subformulas of ϕ is denoted by $\mathtt{Sub}(\phi)$. The *composition size* $|\phi|$ of a formula ϕ is inductively as follows:

$$
\begin{array}{rclrcl}
|\top| & = & 0, & |\phi \wedge \phi'| & = & |\phi| + |\phi'| + 1 \\
|c(x).\phi| & = & 1 + |\phi|, & |\phi \vee \phi'| & = & |\phi| + |\phi'| + 1 \\
|\exists x\ \phi| & = & 1 + |\phi| & & &
\end{array}
$$

Note that this definition implies that query names are of size one.

For all trees t, all valuations $\nu : \mathtt{Var} \to \mathtt{nodes}(t)$ ranging over the nodes of t, and all composition formula $\phi \in \mathcal{C}(L)$ we define the satisfaction relation $t, \nu \models \phi$ as follows:

(i)
$\mathtt{range}(\nu) \subseteq \mathtt{nodes}(t)$

(ii)

$$
\begin{array}{lll}
t, \nu \models \top & & \\
t, \nu[x/_u] \models c(x).\phi & \text{iff} & \left\{ \begin{array}{ll} u \in [\![c]\!](t) & (1) \\ t|_u, \nu \models \phi & (2) \end{array} \right. \\
t, \nu \models \phi_1 \wedge \phi_2 & \text{iff} & t, \nu \models \phi_1 \text{ and } t, \nu \models \phi_2 \\
t, \nu \models \phi_1 \vee \phi_2 & \text{iff} & t, \nu \models \phi_1 \text{ or } t, \nu \models \phi_2 \\
t, \nu \models \exists x\ \phi & \text{iff} & \text{there exists } u \in \mathtt{nodes}(t), \\
 & & \quad \text{s.t. } t, \nu[x/_u] \models \phi
\end{array}
$$

Let us consider the satisfiability of $c(x).\phi$. Condition (1) implies that $[\![c]\!]$ selects the node u in t, condition (2) implies that the interpretation of ϕ is relativized to the subtree of t rooted at u.

Valuations define possible values for free variables in composition formulae. A formula can define an n-ary query by sorting its free variables. Formally, a formula $\phi \in \mathcal{C}(L)$ with free variables $\{x_1, \ldots, x_n\} = \mathrm{FV}(\phi)$ defines the n-ary query $[\![\phi(x_1, \ldots, x_n)]\!]$ such that for all trees t:

$$[\![\phi(x_1, \ldots, x_n)]\!](t) = \{(\nu(x_1), \ldots, \nu(x_n)) \mid t, \nu \models \phi\}$$

4 Examples of Composition Languages

We now discuss some instances of query languages $\mathcal{C}(L)$ by instantiating the parameter L to some concrete

[1] We proved that conjunctions, disjunctions and projections are required to express all MSO-queries by composition of monadic MSO-definable queries

monadic query language.

As first instance, we let L be the monadic query language containing all monadic MSO formulas. For illustration, we consider XML documents defining collections of books, which satisfy the DTD in Fig. 1. Our target is to select all pairs of author names and titles of the same book by composition.

We define the binary query on `firstchild-nextsibling` encodings. Names and titles of a book are contained in siblings of `author`-labeled nodes. To select the pairs, we first select all `author` nodes by the monadic query $[\![c_1]\!]$ defined by the monadic MSO formula $c_1 = \mathtt{lab}_{\mathtt{author}}(x)$. We then compose it with two independant monadic queries $[\![c_2]\!]$ and $[\![c_3]\!]$, for selecting name by $c_2 = \exists y\ \mathtt{root}(y) \wedge \mathtt{child}_1(y,x)$ and title by $c_3 = \exists y\ \mathtt{root}(y) \wedge \mathtt{child}_2(y,x)$. The modeling composition formula is:

$$\phi = \exists z c_1(z).(c_2(x) \wedge c_3(y))$$

Note that according to the DTD and the semantic of composition, the first query can select nodes labeled by `author`, and then, in each subtrees induced by the previous selected nodes, one can select the node labeled by `name`, and the node labeled by `title`, by two independant monadic queries $c'_2 = \mathtt{lab}_{\mathtt{name}}(x)$ and $c'_3 = \mathtt{lab}_{\mathtt{title}}(x)$ respectively. The modeling composition formula is then:

$$\phi = \exists z c_1(z).(c'_2(x) \wedge c'_3(y))$$

A second instance is obtained by composing monadic Datalog queries [8] which are well known to capture all monadic MSO. Indeed, our idea of compositions is very much inspired by the way in which n-ary queries are defined from monadic Datalog queries by the Lixto system for visual Web information extraction [1, 6].

We illustrate the correspondence at the example of selecting pairs of author names and titles of the same books. Such a query is expressed in Lixto by a Monadic Datalog program P and an additional information about the predicate hierarchy, which we model by a tree. We express this query in the `firstchild`−`nextsibling` encodings. The monadic Datalog program P and the predicate hierarchy are given on Figure 4.

We can express the same query by composing the following three monadic Datalog queries by

$$\phi(y,z) = \exists x\, P_1(x).(\ P_2(y)\ \wedge\ P_3(z)\ )$$

.

$$\begin{array}{llll} P_1: & P_{\mathtt{author}}(x) & \text{:-} & \mathtt{lab}_{\mathtt{author}}(x) \\ & & & \text{with the goal } P_{\mathtt{author}} \\ P_2: & P_{\mathtt{name}}(x) & \text{:-} & \mathtt{root}(y), \mathtt{child}_1(y,x) \\ & & & \text{with the goal } P_{\mathtt{name}} \\ P_3: & P_{\mathtt{title}}(x) & \text{:-} & \mathtt{root}(y), \mathtt{child}_2(y,x) \\ & & & \text{with the goal } P_{\mathtt{title}}. \end{array}$$

Implementation We have implemented a rather naive algorithm for answering compositions of monadic queries, defined either in MSO, XPath, or by tree automata. Further monadic query languages are be easily added by new modules called *query machines*. Each monadic query can be expressed by different formalism within the same composition formula.

Our concrete syntax for expressing composition queries is given in Fig. 3. A typical input consists of an XML document and a composition query. The output is an XML document representing the set of all answers. The implementation is done in OCaml.

5 MSO Completeness

We call an n-ary query language MSO-complete if it can express all MSO-definable n-ary queries. For instance, monadic Datalog is known to be a MSO complete monadic query language. In this paragraph, we study the expressiveness of composition languages over MSO-complete monadic query languages.

We show that the composition operator can be expressed in first-order logic, so that n-ary-compositions of monadic MSO definable queries are MSO-definable too.

Let $L = (\mathtt{N}, [\![.]\!])$ be a monadic query language. For every name $c \in \mathtt{N}$ we introduce a binary predicate symbol B_c that we interpret as a binary relation on $B^t_c \subseteq \mathtt{nodes}(t)^2$

$$B^t_c = \{(v,v') \mid v' \in [\![c]\!](t|_v)\}$$

We now consider the first-order logic over the signature $(B_c)_{c \in \mathtt{N}} \cup \mathbb{T}$.

Proposition 1. *Every composition formula $\phi(\bar{x}) \in \mathcal{C}(L)$ is equivalent to some first-order formula $\gamma(\bar{x})$ over the signature $(B_c)_{c \in N} \cup \mathbb{T} \cup \{\lhd^*\}$.*

Proof. We define a function $\langle . \rangle_x$ encoding composition formulas into first-order formulas over the signature $(B_c)_{c \in \mathtt{N}} \cup \mathbb{T} \cup \{\lhd^*\}$ inductively:

$$\begin{array}{lcl} \langle \top \rangle_x & = & \top \\ \langle c(y).\phi \rangle_x & = & B_c(x,y) \wedge \langle \phi \rangle_y \\ \langle \phi_1 \wedge \phi_2 \rangle_x & = & \langle \phi_1 \rangle_x \wedge \langle \phi_2 \rangle_x \\ \langle \phi_1 \vee \phi_2 \rangle_x & = & \langle \phi_1 \rangle_x \vee \langle \phi_2 \rangle_x \\ \langle \exists y\ \phi \rangle_x & = & \exists y\ x \lhd^* y \wedge \langle \phi \rangle_x \end{array}$$

Let $\gamma(x_1, \ldots, x_n) \equiv \exists \mathrm{FV}(\phi) \backslash \{x_1, \ldots, x_n)\ (\exists y\ \mathtt{root}(y) \wedge \langle \phi \rangle_y)$, where $y \notin \mathrm{FV}(\phi)$. Finally note that $\mathtt{root}(x)$ is FO[$\mathbb{T}$]-definable. □

If the monadic query language captures MSO then the binary predicates B_c are MSO-definable. The first important technical contribution of this paper is that the converse holds too.

Theorem 1. *The class of n-ary queries defined by composition of MSO-definable monadic queries is exactly the class of n-ary MSO definable queries.*

To prove the first direction it suffices to show that each predicate B_c is MSO-definable whenever $[\![c]\!]$ is. The

$$\begin{array}{lll} P_{\mathtt{author}}(x) & \text{:-} & \mathtt{lab}_{\mathtt{author}}(x) \\ P_{\mathtt{name}}(x) & \text{:-} & P_{\mathtt{author}}(y),\ \mathtt{child}_1(y,x) \\ P_{\mathtt{title}}(x) & \text{:-} & P_{\mathtt{author}}(y),\ \mathtt{child}_2(y,x) \end{array}$$

(a) program P

$$\begin{tikzcd} & P_{\mathtt{author}} \arrow[dl] \arrow[dr] & \\ P_{\mathtt{name}} & & P_{\mathtt{title}} \end{tikzcd}$$

(b) hierarchy

Figure 2. A set of Monadic Datalog rules and its predicates hierarchy

query	::=	**SELECT** *vars* **FROM** *formula*
formula	::=	*atom* | *formula* **AND** *formula* | *formula* **OR** *formula*
atom	::=	*machine***(***var***)**
vars	::=	*var* | *var,vars*
var	::=	*identifier*
machine	::=	**XPATH[***xpath_specif***]** | **AUTOMATON[***automaton_specif***]** | **MSO[***mso_specif***]**

Figure 3. Concrete sytnax for composition queries

binary MSO formula $\gamma_{B_c}(x,y)$ defining B_c is exactly the formula $\gamma_c(y)$ defining $[\![c]\!]$ where each quantification is relativized to x.

The rest of this section prove the other direction, i.e. the composition of monadic MSO-definable queries is complete for n-ary MSO-definable queries. The proof is based on the equivalence between MSO-definable queries and node selection automata as defined in [16], which is a consequence of the seminal theorem of Thatcher and Wright [17].

We recall that a *node selection automaton* (NSA) is a pair (A,S) where $A=(\Sigma,Q,F,\Delta)$ is a tree automata and S is a set of selection tuples $\overline{q}$. We write $(A,\overline{q})$ instead of $(A,\{\overline{q}\})$. A *run* of a tree automata A over a tree t is a tree r isomorphic to t via an isomorphism Φ, where each node is labeled in Q, and such that the following holds:

- if $v \in \mathtt{nodes}(t)$ is a leaf labeled by $a \in \Sigma$, then $a \to \mathtt{lab}_r(\Phi(v))$ is in Δ,
- if $v \in \mathtt{nodes}(t)$ is an inner node labeled by $f \in \Sigma$, and $v_1, v_2 \in \mathtt{nodes}(t)$ are its first child and its second child respectively, then the rule $f(\mathtt{lab}_r(\Phi(v_1)), \mathtt{lab}_r(\Phi(v_2))) \to \mathtt{lab}_r(\Phi(v))$ is in Δ.

A run r of A over t is successful iff its root is labeled by an accepting state from F. A NSA (A,S) selects a tuple of nodes $(v_1,\ldots,v_n)$ of a tree t iff there exists a successful run r over t (isomorphic to t via Φ), and a selection tuple $(q_1,\ldots,q_n) \in S$, such that for each $i \in \{1,\ldots,n\}$, the node $\Phi(v_i)$ is labeled by q_i in r. When it is clear from the context we will omit the isomorphism Φ. Finally, the class of MSO-definable n-ary queries is exactly the class of n-ary queries defined by node selection automata over binary trees [17, 16].

In order to prove Theorem 1 we introduce some notations. Given a set R, an n-tuple $\overline{r}=(r_1,\ldots,r_n) \in R^n$, and a set $J \subseteq \{1,\ldots,n\}$, we denote by $\Pi_J(\overline{r})$ the projection of $\overline{r}$ w.r.t. J, defined by $\Pi_J(\overline{r}) = (r_i)_{i\in J}$. In particular, $\Pi_\emptyset(\overline{r}) = ()$. Given a tree t, an n-tuple of nodes $\overline{v}=(v_1,\ldots,v_n) \in \mathtt{nodes}(t)^n$, a NSA $(A,\overline{q})$ with a selection tuple $\overline{q}=(q_1,\ldots,q_n) \in Q^n$, and a state $q \in Q$, a *q-run of $(A,\overline{q})$ over t selecting $\overline{v}$* is a run of A over t such that the root is labeled by q, and v_i is labeled by q_i for each $i \in \{1,\ldots,n\}$. In particular, when $n=0$, a q-run of $(A,())$ over t selecting the empty sequence is a run of A over t labeling the root by q.

Lemma 1. *Let $n \geq 2$ be a natural. Let $t \in T_\Sigma$ be a binary tree. Let $\overline{v}=(v_1,\ldots,v_n)$ be a tuple of length n of nodes from t, such that there exists at least two different nodes. Let v_a be the least common ancestor of $\overline{v}$. Let v_a^1 be the first child of v_a, and v_a^2 its second child. Define I,J,K as follows:*

$$\begin{array}{lll} I & = & \{i \mid v_a = v_i\} \\ J & = & \{j \mid v_a^1 \lhd^* v_j\} \\ K & = & \{k \mid v_a^2 \lhd^* v_k\} \end{array}$$

Let $(A,\overline{q})$ be a NSA, and q a state, then there exists a q-run r of A over t selecting $\overline{v}$ iff

$\exists q', q'' \in Q$ s.t.
- there exists a q-run of $(A,\Pi_I(\overline{q}))$ over t selecting $\Pi_I(\overline{v})$ and labeling v_a^1, v_a^2 by q', q'' respectively
- there exists a q'-run of $(A,\Pi_J(\overline{q}))$ over $t|_{v_a^1}$ selecting $\Pi_J(\overline{v})$
- there exists a q''-run of $(A,\Pi_K(\overline{q}))$ over $t|_{v_a^2}$ selecting $\Pi_K(\overline{v})$

Proof. The proof is not difficult and left to the reader. $\square$

Lemma 2. *Let n be a natural. Given a node selection automaton $(A,\overline{q})$ where $\overline{q}$ is an n-tuple of states, given a state $q \in Q$, there exists a composition formula $\phi_{A,\overline{q},q}(x_1,\ldots,x_n)$ over MSO-definable monadic queries such that for all Σ-tree t, for all $\overline{v} \in \mathtt{nodes}(t)^n$, the following are equivalent:*

(i) *there exists a q-run of A over t selecting $\overline{v}$*
(ii) $\overline{v} \in [\![\phi_{A,\overline{q},q}(x_1,\ldots,x_n)]\!](t)$

Proof. We construct the formula inductively on n. The construction mimics the decomposition given by lemma 1.
If $n = 0$ we take $\phi_{A,\overline{q},q} = \exists x\, c_0(x)$ where $[\![c_0]\!](t)$ is equal to $\mathtt{nodes}(t)$ if and only if there exists a q-run of A over t. By Thatcher and Wright's theorem, this monadic query is MSO-definable.
If $n = 1$, then $\overline{q} = (p)$ for some $p \in Q$, and we take $\phi_{A,(p),q}(x) = c_1(x)$ where $[\![c_1]\!]$ is defined by the NSA $(A,(p))$. Again by Thatcher and Wright's theorem, this query is MSO-definable.
If $n > 1$, we consider two cases depending on whether the variables $x_1,\dots,x_n$ will be instantiated by the same node or not. So $\phi_{A,\overline{q},q}(x_1,\dots,x_n)$ will be written as a disjunction $\phi^{eq}_{A,\overline{q},q} \vee \phi^{neq}_{A,\overline{q},q}$:

- case 1 (variables will be instantiated by the same node). Let $\gamma_{(A,\overline{q})}(\overline{x})$ be an MSO formula such that for a tree t and an n-tuple $\overline{v}$ of nodes of t, it holds that $\overline{v} \in [\![\gamma_{(A,\overline{q})}]\!](t)$ iff there exists a q-run of $(A,\overline{q})$ over t selecting $\overline{v}$. It is easy to show that this formula exists, by Thatcher and Wright's theorem. Then we take $\phi^{eq}_{A,\overline{q},q}(x_1,\dots,x_n) = \exists x\ \ c_1(x).(\bigwedge_i c_r(x_i))$, where $[\![c_1]\!]$ is the query defined by the monadic MSO formula $\exists y_1,\dots,y_{n-1} \bigwedge_i (y_n = y_i) \wedge \gamma_{(A,\overline{q})}(y_1,\dots,y_n)$ and $[\![c_r]\!](t)$ selects the root of t, for any tree t.

- case 2 (variables will be instantiated by at least two different nodes). Let $\overline{x}$ denotes $(x_1,\dots,x_n)$ and let $\mathcal{P}_n$ be the sets of partitions (with possibly empty parts) of $\{1,\dots,n\}$ such that for each partition, there exists at most one empty part. We define $\phi^{neq}_{A,\overline{q},q}(\overline{x})$ by:

$$\begin{array}{l} \bigvee_{\{I,J,K\}\in\mathcal{P}_n} \bigvee_{q',q''\in Q} \\ \quad \exists x \exists y \exists z\ c^{q}_{q',q''}(x). \\ \qquad (\bigwedge_{i\in I} c_r(x_i) \\ \qquad \wedge c_1(y).\phi_{A,\Pi_J(\overline{q}),q'}(\Pi_J(\overline{x})) \\ \qquad \wedge c_2(z).\phi_{A,\Pi_K(\overline{q}),q''}(\Pi_K(\overline{x}))) \end{array}$$

  where $[\![c_1]\!](t)$ selects the first child of the root of t, and $[\![c_2]\!](t)$ its second child. For any tree t, the query $[\![c^{q}_{q',q''}]\!](t)$ selects a node $v \in \mathtt{nodes}(t)$ iff there exists a q-run of $(A,\Pi_I(\overline{q}))$ over t selecting $(v,v,\dots,v)$ (of length $|I|$), such that its first child is labeled by q', and its second child by q''. This query is MSO-definable, again by Thatcher and Wright's theorem. Remark that subformulae $\phi_{A,\Pi_J(\overline{q}),q'}(\Pi_J(\overline{x}))$ and $\phi_{A,\Pi_K(\overline{q}),q'}(\Pi_K(\overline{x}))$ are recursively well defined, since $|K|,|J| < n$.

  The rest of the proof is a direct application of Lemma 1.

$\square$

To conclude the proof of Theorem 1, we state the following corollary:

Corollary 1. *For each MSO formula $\gamma(\overline{x})$, there exists an equivalent composition formula $\phi(\overline{x})$ over MSO-definable monadic queries.*

Proof. By [17, 16], there exists a NSA (A,S) equivalent to γ, and we define ϕ by: $\phi = \bigvee_{\overline{q}\in S} \bigvee_{q\in F} \phi_{A,\overline{q},q}$, where $\phi_{A,\overline{q},q}$ has been defined in the previous lemma. $\square$

6 Algorithmic Complexity

In this paragraph $L = (\mathtt{N}, [\![.]\!])$ is a monadic query language, and we suppose that there exists an algorithm for the model-checking problem in time-complexity $\mathtt{mc}(c,t)$, where $c \in \mathtt{N}$ and $t \in T_\Sigma$, and an algorithm for the query answering problem in time-complexity $\mathtt{qa}(c,t)$.

Fig. 4 represents a simple algorithm for the model-checking problem of a formula ϕ, a tree t and a valuation ν. It is written in a pseudo ML-like code. It runs in time $O(|\phi|M|t|^{\max_{\phi'\in\mathtt{Sub}(\phi)}(|FV(\phi')|)} + |\phi|^2)$ where $M = \max_{c(x)\in\mathtt{Sub}(\phi)} \mathtt{mc}(c,t)$.

This gives a naive algorithm for the query answering problem: generate all the valuations of free variables of a formula ϕ in a tree t, and apply the model-checking algorithm on them. This leads to an exponential grow up, but it is not clear how to avoid it since the satisfiability problem of monadic query composition is NP-hard.

Proposition 2. *Let $\Sigma = \{0,1,\circ\}$ be an alphabet and $L = (\{c_0,c_1\}, [\![.]\!])$ a monadic query language over Σ where $[\![c_b]\!]$ selects all the nodes labeled by $b \in \{0,1\}$ in Σ-trees. Let t the binary whose roots is labeled by $\circ$, its first child by* 0*, and its second child by* 1*. Given a composition formula ϕ over L, the satisfiability problem of ϕ over t is NP-hard.*

Proof. To prove that it is NP-hard we give a polynomial reduction of CNF satisfiability into our problem. The idea is to associate with a given CNF formula $\Psi = \bigwedge_{1\le i\le p} C_i$ a composition formula $\phi = \bigwedge_{1\le i\le p} \phi_i$ over L. Each ϕ_i is a composition formula associated to the i-th clause C_i. It is defined by associating to each litteral x_j the atomic formula $c_1(x_j)$ and to $\neg x_j$ the formula $c_0(x_j)$, and to a disjunction of litterals a disjunction of atomic formulae. For example, if we consider $\Psi = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$, then $\phi = (c_1(x_1) \vee c_0(x_2)) \wedge (c_1(x_2) \vee c_0(x_3))$. $\square$

Composition and conjunctive queries Conjunctive queries over finite relational structures have been widely studied by the database community since it is the most common database query in practice. The particular case of conjunctive queries over unranked trees have been studied in [7] over particular binary XPath axis $\mathcal{A}$ = { Child, Child$^+$, Child*, NextSibling,

$$
\begin{array}{lcl}
\textbf{let}\ \mathtt{check}(\phi,t,\nu) & = & \textbf{match}\ \phi\ \textbf{with} \\
 & & \mid \top \ \rightarrow\ \text{true} \\
 & & \mid \psi(x).\phi' \ \rightarrow\ \nu(x) \in [\![\psi]\!](t)\ \wedge\ \mathtt{check}(\phi', t|_{\nu(x)}, \nu)\ \wedge \forall y \in \mathrm{FV}(\phi')\ \nu(x) \lhd^* \nu(y) \\
 & & \mid \phi' \vee \phi'' \ \rightarrow \mathtt{check}(\phi',t,\nu)\ \vee \mathtt{check}(\phi'',t,\nu) \\
 & & \mid \phi' \wedge \phi'' \ \rightarrow \mathtt{check}(\phi',t,\nu)\ \wedge \mathtt{check}(\phi'',t,\nu) \\
 & & \mid \exists x\ \phi' \ \rightarrow \bigvee_{u \in \mathtt{nodes}(t)} \mathtt{check}(\phi',t,\nu[x/_u])
\end{array}
$$

Figure 4. Model-checking algorithm for a formula $\phi \in \mathcal{C}(L)$, a tree t and a valuation ν

NextSibling$^+$, NextSibling*, Following }. Surprisingly the complexity of these queries quickly fall into NP-hardness. Since each conjunctive queries over these axis in an unranked tree is expressible by a composition query over a particular monadic query language in binary trees, all the complexity lower bounds from [7] apply to our formalism. For example, the satisfiability problem of a composition query over the monadic query langue $(\{c_{1^*}, c_2\}, [\![.]\!])$ is NP-hard w.r.t. combined complexity, where $[\![c_{1^*}]\!](t) = \{v \mid \text{Child}_1^*(\mathtt{root}(t), v)\}$ and $[\![c_2]\!](t) = \{v \mid \text{Child}_2(\mathtt{root}(t), v)\}$.

In the next section we propose a composition fragment for which the satisfiability problem is in PTIME whenever this holds for the underlying monadic query language, and give an efficient algorithm for query answering. In addition we prove that this fragment can express all MSO-definable *n*-ary queries whenever the underlying monadic query language captures MSO.

7 An MSO-Complete and Tractable Fragment

In this section, we introduce a “tractable” syntactic fragment of composition formulae $\mathcal{E}(L)$, that leads to an *n*-ary MSO-complete query language (as soon as the monadic query language L is), while enjoying efficient query answering algorithms.

Let L be a language of MSO-definable monadic queries. In this fragment, variable sharing between conjunctions and composition are not permitted, more precisely, if $\phi \wedge \phi'$ and $c(x).\phi''$ are $\mathcal{E}(L)$-formula, then $\mathrm{FV}(\phi) \cap \mathrm{FV}(\phi') = \emptyset$, and $x \notin \mathrm{FV}(\phi'')$. CDuce patterns for instance are built under this restriction for conjunctions [4].

If the satisfiability problem for the underlying query language is PTIME, then it holds for the composition fragment too. The algorithm is based on dynamic programming – a satisfiability table defined inductively is computed with memoization –. Then the query answering algorithm processes the formula inductively under the assumption that it is satisfied in the current tree.

7.1 MSO-completeness

We start by a theorem on expressiveness of the fragment $\mathcal{E}(L)$, over MSO-definable monadic queries.

Theorem 2. *Let L be a language of MSO-definable monadic queries. The class of n-ary queries defined by $\mathcal{E}(L)$-formulae is exactly the class of n-ary MSO-definable queries.*

Proof. The proof is the same than those of Theorem 1. It suffices to remark that the constrution of an equivalent composition formula given in Theorem 1 respects the required restrictions on variable sharing. □

7.2 Answering algorithm

In this section we give an algorithm for answering a composition query q on a tree t, so that the complexity may depend on the size of the output. Since the answering complexity depends on the maximal number of free variables of the subformulae of the formula defining the query, we first show that each composition formula ϕ is equivalent to a composition formula where there is a most 1 free variable different from the free variables of ϕ in its subformulae (wlog we assume that the quantified variables of ϕ are different from the free variables of ϕ). Moreover, in order to avoid the problem of non-valued variables – for example in the formula $c(x) \vee c(y)$ –, we complete each formula so that each part of disjunctions has the same free variable sets. For instance the formula $c(x) \vee c(y)$ is rewriting into the equivalent formula $(c(x) \wedge \text{true}(y)) \vee (\text{true}(x) \wedge c(y))$. The size of the output formula can be at most quadratic in the size of the input formula.

Let $L = (\mathtt{N}, [\![.]\!])$ be a monadic query language. Let $t \in T_\Sigma$ be a tree and let $\phi \in \mathcal{E}(L)$ a composition formula. We suppose to have an algorithm to answer monadic queries. The query answering algorithm processes in four steps:

1. rewrite ϕ into an equivalent formula ϕ' in which there is at most one free variable different from the free variables of ϕ', in its subformulae, and such that for each $\gamma \vee \gamma' \in \mathtt{Sub}(\phi')$, $\mathrm{FV}(\gamma) = \mathrm{FV}(\gamma')$;
2. compute two data structures $Q_a : \mathtt{N} \times \mathtt{nodes}(t) \rightarrow \mathtt{nodes}(t)$ and $Q_c : \mathtt{N} \times \mathtt{nodes}(t) \times \mathtt{nodes}(t) \rightarrow \{0,1\}$ such that given a query name $c \in \mathtt{N}$ appearing in ϕ', and two nodes $v, v' \in \mathtt{nodes}(t)$, $Q_a(c,v)$ returns the set $\{v' : v' \in [\![c]\!](t|_v)\}$ in linear time in the size of the output, and $Q_c(c,v,v')$ checks in constant time whether $v' \in [\![c]\!](t|_v)$;
3. compute a data structure $\mathtt{Sat} : \mathtt{Sub}(\phi') \times \mathtt{nodes}(t) \rightarrow \{0,1\}$ such that $\mathtt{Sat}(\phi'', v)$ checks in

constant time whether a formula $\phi'' \in \mathtt{Sub}(\phi')$ is satisfied in $t|_v$;

4. answer the query by processing the formula ϕ' recursively with satisfiability tests, doubles elimination, and memoization.

Step 1 Let ϕ be a composition formula. Wlog assume that quantified variables of ϕ are different from its free variables. We define the width $\mathtt{w}(\phi)$ of ϕ as the maximal number, over the subformulae of ϕ, of free variables different from the free variables of ϕ. More formally $\mathtt{w}(\phi) = \mathtt{max}_{\phi' \in \mathtt{Sub}(\phi)} |\mathrm{FV}(\phi') \backslash \mathrm{FV}(\phi)|$. As we said we transform ϕ into an equivalent formula ϕ' with $\mathtt{w}(\phi') \leq 1$. The transformation is simple by pushing down the quantifiers. We sum up it in the following lemma:

Lemma 3. *Each query q defined by a composition formula $\phi \in \mathcal{E}(L)$ is equal to some query defined by a composition formula $\phi' \in \mathcal{E}(L)$ such that $\mathtt{w}(\phi') \leq 1$.*

Proof. We define the translation of ϕ into ϕ' by the following rewriting rules:

$$\begin{array}{lcl}
\exists x\ (\gamma \vee \gamma') & \rightarrow & (\exists x\ \gamma) \vee (\exists x\ \gamma') \\
\exists x\ (\gamma \wedge \gamma') & \rightarrow & (\exists x\ \gamma)\ \wedge\ (\exists x\ \gamma') \\
\exists x\ c(y).\phi & \rightarrow & c(y).(\exists x\ \phi) \text{ with } y \neq x \\
\exists x\ \gamma & \rightarrow & \gamma \text{ if } x \notin \mathrm{FV}(\gamma)
\end{array}$$

We can show this rewriting system to terminate, and to be confluent. The normal form is a formula where each occurence of a quantified variable in an atomic formula $c(x)$ is preceded by an existential quantification $\exists x\ c(x)$. Hence, normal forms are of width at most 1. Now we show that the normal form ϕ' of a formula ϕ is equivalent to ϕ. The only difficulties come from $\exists x\ (\gamma \wedge \gamma') \rightarrow (\exists x\ \gamma)\ \wedge\ (\exists x\ \gamma')$ and $(\exists x\ c(y).\phi) \rightarrow (c(y).(\exists y\ \phi)$. The first case holds since $\mathrm{FV}(\gamma) \cap \mathrm{FV}(\gamma') = \emptyset$, and the following proves the second case:
$t, \nu[y/_{v'}] \models (\exists x\ c(y).\phi)$
iff there exists $v \in \mathtt{nodes}(t)$ s.t. $t, \nu[y/_{v'}][x/_v] \models c(y).\phi$
iff there exists $v \in \mathtt{nodes}(t|_{v'})$, $v' \in [\![c]\!](t)$ and $t|_{v'}, \nu[x/_v] \models \phi$
iff $t, \nu[y/_{v'}] \models c(y).(\exists x\ \phi)$.
We conclude by induction on the reduction length. ☐

Remark that the size of the resulting formula is linear – multiply by two – in the size of the input formula, since each occurence of free variable is preceded by its quantification. Then we transform ϕ' so that each part of a disjunction shares the same free variable sets, and such that each quantified variable is different from each free variable of ϕ'.

Step 2 It is quite obvious, by using hash tables.

Step 3 We compute – using memoization – a table $\mathtt{Sat}[.,.]$ defined inductively by:

$$\begin{array}{rcll}
\mathtt{Sat}[\top, u] & = & 1 & (1) \\
\mathtt{Sat}[c(x).\phi, u] & = & \bigvee_{u' \in Q_a(c,u)} \mathtt{Sat}[\phi, u'] & (2) \\
\mathtt{Sat}[\phi \wedge \phi', u] & = & \mathtt{Sat}[\phi, u] \wedge \mathtt{Sat}[\phi', u] & (3) \\
\mathtt{Sat}[\phi \vee \phi', u] & = & \mathtt{Sat}[\phi, u] \vee \mathtt{Sat}[\phi', u] & (4) \\
\mathtt{Sat}[\exists x\ \phi, u] & = & \mathtt{Sat}[\phi, u] & (5)
\end{array}$$

Step 4 The last phase is given on Fig. 5. Moreover, we use memoization to avoid exponential grow-up. Valuations are represented by sequences of pairs (variable, node). We assume union and projection operations to eliminate doubles, so that their time complexities are linear in the input sets. This can be done by storing tuples in hash tables.

7.3 Answering Complexity

In this section we study the complexity of the previous algorithm. Let $L = (\mathtt{N}, [\![.]\!])$ be a monadic query language. Inputs of the algorithm are a tree t and a composition formula $\phi \in \mathcal{E}(L)$. Moreover, we suppose to have of an algorithm to answer $[\![c]\!]$ on a tree t, for each $c \in \mathtt{N}$, in time complexity $\mathtt{qa}(n,t)$. We write $M(\phi,t)$ for $\max_{v \in \mathtt{nodes}(t), c(x) \in \mathtt{Sub}(\phi)} \mathtt{qa}(c, t|_v)$. We sum-up the complexity by the following proposition:

Proposition 3. *Answering a query q defined by a composition formula $\phi \in \mathcal{E}(L)$ is in time $O(M(\phi,t)|t||\phi| + |\phi|^2|t|^2|\phi(t)|)$, where $|\phi(t)|$ is the output size.*

Proof. The first step produces a formula ϕ' such that $|\phi'| = O(|\phi|^2)$. The second step is in time $O(M(\phi,t)|t||\phi|)$, and the computation of the satisfiability table is in time $O(|\phi'||t|^2) = O(|\phi|^2|t|^2)$.
It remains to show the time complexity of algorithm depicted in figure 5 to be $O(|\phi'||t|^2 nK)$, where K is the number of solutions and n the arity of the query – we consider that $|\phi(t)| = Kn$ –. We are going to show that each recursive call returns at most $|t|K$ valuations, and performs at most $O(|t| + nK|t|)$ operations. Each call to $\mathtt{ans}$ begins by a satisfiability test, so that the following property holds: if $\mathtt{ans}(\gamma, t, v)$ is a recursive call occuring during the processing of ϕ', then the projection of each valuation returned by $\mathtt{ans}(\gamma, t, v)$ on the variables from $\mathrm{FV}(\phi')$ can be extended to a valuation ν such that $t, \nu, \mathtt{root}(v) \models \phi'$. Hence, the number of valuations returned by $\mathtt{ans}(\gamma, t, v)$ is at most $|t|^{|\mathrm{FV}(\gamma) \backslash \mathrm{FV}(\phi')|}K$. Moreover, since $\mathtt{w}(\phi') = 1$, we get $|\mathrm{FV}(\gamma) \backslash \mathrm{FV}(\phi')| \leq 1$. It is clear that for conjunctions, disjunctions, and projections, each recursive call performs at most $O(|t|nK)$ operations. If γ is of the form $c(x).\gamma'$, then $\mathrm{FV}(\gamma') = \mathrm{FV}(\gamma') \cap \mathrm{FV}(\phi')$, since $\mathtt{w}(\gamma) = 1$. Hence, any recursive call to $\mathtt{ans}(\gamma', t, v')$ for $v' \in [\![c]\!](t)$ returns at most K valuations. Moreover, there are at most $|t|$ nodes satisfying $[\![c]\!](t)$, so that the recursive call $\mathtt{ans}(\gamma, t, v)$ performs at most $|t| + nK|t|$ operations.
Finally, since we use memoization, there are at most $|t||\phi'|$ recursive call to $\mathtt{ans}$, so that the whole complexity of $\mathtt{ans}$ on input, ϕ', t and $\mathtt{root}(t)$ is $O(|\phi'||t|^2 nK))$. ☐

8 Conclusion

8.1 Summary.

We proposed and investigated an n-ary query language $\mathcal{C}(L)$ in which queries are specified as composition of monadic queries. The choice of the underlying monadic query language L is parametric, so that we can express a wide variety of n-ary query specification languages, for

$$
\begin{array}{lll}
\textbf{let}\ \texttt{ans}(\phi,t,u) & = & \textbf{if}\ \texttt{Sat}[\phi,u]\ \textbf{then} \\
 & & \textbf{match}\ \phi\ \textbf{with} \\
 & & \quad \mid \top \ \rightarrow\ \{\varepsilon\} \\
 & & \quad \mid c(x).\phi' \rightarrow\ \bigcup_{u' \in Q_a(c,u)} \{(x,u')\cdot\nu \mid \nu \in \texttt{ans}(\phi',t,u')\} \\
 & & \quad \mid \phi' \wedge \phi'' \ \rightarrow\ \texttt{ans}(\phi',t,u) \times \texttt{ans}(\phi'',t,u) \\
 & & \quad \mid \phi' \vee \phi'' \ \rightarrow\ \texttt{ans}(\phi',t,u) \cup \texttt{ans}(\phi'',t,u) \\
 & & \quad \mid \exists x\phi \ \rightarrow\ \{\nu\ :\ \texttt{dom}(\nu) = \texttt{dom}(\nu')\backslash x, \nu = \nu'|_{\texttt{dom}(\nu)\backslash x}, \nu' \in \texttt{ans}(\phi,t,u)\} \\
 & & \textbf{else}\ \emptyset \\
\quad \textbf{in} & & \\
\quad \texttt{ans}(\phi,t,\texttt{root}(t)) & &
\end{array}
$$

Figure 5. Answering algorithm with implicit memoization

instance composition of XPath formula, Monadic Datalog programs or node selection automata. We proved our language to capture MSO as soon as the underlying monadic query language capture MSO too. We proved the satisfiability problem to be NP-hard and proposed an efficient fragment $\mathcal{E}(L)$ of the composition language which remains MSO-complete as soon as L captures MSO. We gave an algorithm for the query answering problem in time $O(M(\phi,t)|t||\phi| + |\phi|^2|t|^2|\phi(t)|)$, where $|\phi(t)|$ is the output size and $M(\phi,t)$ is the maximal complexity of the query answering problem over subtrees of t, of the monadic queries appearing in ϕ.

8.2 Future Work.

A more practical aspect is the extension of the existing implementation of query composition to the algorithms in *Section 7* and the comparison of their query answering efficiencies with other querying languages, such as implementations of XQuery, and programming languages such as ℂDuce .

We would like to investigate the correspondence – mentioned in *Section 4* between the underlying query formalism of Lixto and our query composition language over Monadic Datalog programs. In particular, we think that there exists a systematic translation between the two formalisms.

Finally, in some cases it seems to be more efficient to have the possibility to navigate everywhere in the tree, without restriction on subtrees. The binary query example given in *Section 3*, on the tree of figure 1 seems to be more natural when one first selects a node labeled by `name`, and then its sibling. In this way it is interesting to investigate the more general problem of binary query composition.

We would like to thank Manuel Loth who worked on the implementation of monadic query composition.

9 References

[1] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *28th International Conference on Very Large Data Bases*, pages 119–128, 2001.

[2] Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. Cduce: an XML-centric general-purpose language. *ACM SIGPLAN Notices*, 38(9):51–63, 2003.

[3] Julien Carme, Aurlien Lemay, and Joachim Niehren. Learning node selecting tree transducer from completely annotated examples. In *7th International Colloquium on Grammatical Inference*, volume 3264 of *Lecture Notes in Artificial Intelligence*, pages 91–102. Springer Verlag, 2004.

[4] Giuseppe Castagna. Patterns and types for querying XML. In *10th International Symposium on Database Programming Languages*, Lecture Notes in Computer Science. Springer Verlag, August 2005.

[5] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proc. LICS '02: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 215–224, Washington, DC, USA, 2002. IEEE Computer Society.

[6] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto data extraction project - back and forth between theory and practice. In *23rd ACM SIGPLAN-SIGACT Symposium on Principles of Database Systems*, pages 1–12. ACM-Press, 2004.

[7] G. Gottlob, C. Koch, and K. Schulz. Conjunctive queries over trees, 2004.

[8] Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. In *21rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 17–28. ACM-Press, 2002.

[9] Georg Gottlob and Christoph Koch. Monadic queries over tree-structured data. In *17th Annual IEEE Symposium on Logic in Computer Science*, pages 189–202, Copenhagen, 2002.

[10] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing xpath queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005.

[11] Haruo Hosoya and Benjamin Pierce. Regular expression pattern matching for XML. *Journal of Functional Programming*, 6(13):961–1004, 2003.

[12] Leonid Libkin. Logics over unranked trees: an

overview. In *Automata, Languages and Programming: 32nd International Colloquium*, number 3580 in Lecture Notes in Computer Science, pages 35–50. Springer Verlag, 2005.

[13] Frank Neven and Jan Van Den Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, 49(1):56–100, 2002.

[14] Frank Neven and Thomas Schwentick. Query automata. In *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*, pages 205–214, 1999.

[15] Frank Neven and Thomas Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.

[16] Joachim Niehren, Laurent Planque, Jean-Marc Talbot, and Sophie Tison. N-ary queries by tree automata. In *10th International Symposium on Database Programming Languages*, volume 3774 of *Lecture Notes in Computer Science*, pages 217–231. Springer Verlag, September 2005.

[17] J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.