N-ary Queries by Tree Automata

Joachim Niehren Laurent Planque Jean-Marc Talbot Sophie Tison

INRIA Futurs, LIFL, Lille, France
`www.grappa.univ-lille3.fr/mostrare`

N-ary queries in trees select sets of n-tuples of nodes. We propose and investigate representation formalisms for n-ary queries by tree automata, both for ranked and unranked trees. We show that existential run-based queries capture MSO in the n-ary case, as well as universal run-based queries. We then characterize queries by runs of unambiguous tree automata, and show how to decide whether an MSO defined query belongs to this class.

Keywords: Trees, logic, automata, semi-structured documents, information extraction.

1 Introduction

The problem of selecting nodes in trees is the most basic and widespread database querying problem in the context of XML [18, 11, 9]. This is the problem to define monadic node queries in trees. Many applications of semi-structured documents, however, need to solve more complex querying problems. The next more general problem is to define *n-tuples of nodes* in trees, i.e., to express n-ary queries. The W3C standards XQuery and XSLT generalize queries further into tree transformations [8, 24].

N-ary queries in unranked trees serve as *wrappers* in information extraction from semi-structured documents [1]. A typical problem here is to extract all pairs of products and prices from an HTML or XML document. It can be reduced to find a binary query in trees that distinguishes those leaves with the required content. Such queries can either be programmed manually or visually [10], or be inferred automatically from annotated examples [13, 5].

Monadic second-order logic (MSO) is the most traditional representation language for regular n-ary queries in trees [23, 20]. Every MSO formula with n free first-order variables defines an n-ary query. MSO is highly expressive, succinct, and robust under many wishful operations. On the other hand side, its usage remains limited due to its high computational complexity in query answering.

In the case of monadic queries, a number of alternative logical query languages were investigated. The W3C standard XPATH, for instance, can express monadic queries that are first-order [14]. *Monadic Datalog* is the logic programming approach. It is advantageous because of its high expressiveness (all monadic MSO queries in trees can be specified), efficient linear time combined complexity for query answering, and its appropiateness for visual wrapper specification. Monadic Datalog in unranked trees underlies the Lixto system [1] for Web information extraction. Lixto indeed supports n-ary queries for information extraction by composing monadic queries for all slots.

In this paper, we follow the tree automaton approach to express n-ary queries in trees. Tree automata are known to have the same querying power than MSO by Thatcher and Wright's 1968 theorem [23]. We follow recent ideas to define n-ary queries by successful runs of tree automata [16, 9, 2]. Previous approaches, however, either remain limited to monadic queries [9] or rely on more complex automata devices, such as pushdown forest automata [2] and attribute grammars [16] (BAGs for monadic queries and RAGs for n-ary queries). None of these formalisms has been proved to capture n-ary MSO definable queries so far. The only formalism we conjecture to do so is that of Seidl and Berlea [2].

The situation for unranked trees is unsatisfying in that numerous more recent automata notions exist to which standard results do not immediately apply. Hedge automata [4] are most popular, beyond forest automata [15] and query automata [18]. Selection automata [9] and stepwise tree automata [6] improve on that situation; they operate as usual but on (different) binary encodings of unranked trees.

In this paper, we propose and investigate representation formalisms for n-ary queries by standard tree automata, both for ranked and unranked trees. We represent n-ary queries by successful runs of tree automata. We show that existential run-based queries capture the class of MSO definable queries, as well as universal run-based queries. Similar results were known for monadic queries but are new for n-ary queries.

We then investigate the querying power of unambiguous tree automata. They subsume deterministic automata while limiting the amount of nondeterminism. Monadic run-based queries by unambiguous tree automata are known to capture monadic MSO definable queries in contrast to deterministic tree automata. (These are the IBAGs of [16].) For the n-ary case, we prove that run-based queries by unambiguous are strictly less expressive than MSO. They capture only finite unions of Cartesian closed regular queries. This is the class of n-ary queries that can be defined by disjunctions of conjunctions of MSO formulas with one free variable each. We show that it is decidable whether an MSO defined query belongs to that restricted class.

Representing n-ary queries by tree automata is advantageous for query induction by methods from grammatical inference [13]. Query induction is important for improving visual wrapper induction, as argued by Gottlob et. al. [12]. Recent induction methods for monadic queries indeed rely on run-based queries by unambiguous tree automata [5]. The results of this paper clarify the principal limitations of this approach.

2 Regular queries and MSO

We recall known results on MSO definable n-ary queries [22, 17, 20]. We develop our theory for binary trees in a first step. This will be sufficient to deal with unranked trees in a second step (Section 5).

2.1 N-ary queries in binary trees

We start from a finite signature Σ of binary function symbols f and constants a. The meta variable c ranges over arbitrary symbols in Σ. A *binary tree* $t \in T_\Sigma$ is a ground term over Σ.

A *node* π of a tree t is a word over the alphabet $\{1, 2\}$ that addresses some subtree of t from the root of t. We write $\mathsf{nodes}(t)$ for the set of nodes of t. The empty word ϵ

is the root of t. We write $\pi \cdot \pi'$ for the concatenation of the words π and π'. The node $\pi \cdot 1$ of a tree t is the *first child* of the node π in t, while $\pi \cdot 2$ is its *second child*. A *leaf* is a node without children. An *inner node* of a tree is a node that is not a leaf. We will freely identify trees t over Σ with *labeling functions* of type $t : \mathsf{nodes}(t) \to \Sigma$, such that for all $a, f \in \Sigma$, $t_1, t_2 \in T_\Sigma$, and $i \cdot \pi \in \{1,2\}^*$ (where i is the word 1 or 2) :

$$a(\epsilon) = a, \qquad f(t_1,t_2)(\epsilon) = f, \qquad f(t_1,t_2)(i \cdot \pi) = t_i(\pi) \quad \text{if } \pi \in \mathsf{nodes}(t_i)$$

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 sets of n-tuples of nodes in t:*

$$\forall t \in T_\Sigma : q(t) \subseteq \mathsf{nodes}(t)^n$$

Simple examples for monadic queries in binary trees over Σ are the functions leaf and root that map trees t to the sets of their leaves resp. to the singleton $\{\epsilon\}$. The monadic queries label_c for symbols $c \in \Sigma$ map trees t to the set of c-labeled nodes π of t, i.e., $\pi \in \mathsf{label}_c(t)$ iff $t(\pi) = c$. The binary query $\mathsf{first\_child}$ relates nodes π to their first child $\pi \cdot 1$ if it exists, while the query $\mathsf{next\_sibl}$ relates first children $\pi \cdot 1$ to their next sibling to the right $\pi \cdot 2$.

Our definition of n-ary queries is quite general in that it does not exclude non-regular queries. For instance, we can query for all pairs (π, π') in trees t such that the subtrees of t on below of π and π' are equal in structure. This query can indeed be expressed by the RAG's of Neven and Bussche [16].

2.2 MSO definable queries

We introduce n-ary queries definable in monadic second-order logic (MSO) in trees which captures all regular queries.

In MSO, binary trees $t \in T_\Sigma$ are seen as *logical structures* that we equally denote by t. The domain of this structure is the set $\mathsf{nodes}(t)$. Its signature consists of the binary relation symbols $\mathsf{first\_child}$ and $\mathsf{next\_sibl}$ and the monadic relation symbols label_c for all $c \in \Sigma$. These symbols are interpreted by the corresponding node relations of t.

$$\begin{aligned} \mathsf{first\_child}^t &= \{(\pi, \pi \cdot 1) \mid \pi \cdot 1 \in \mathsf{nodes}(t)\} \\ \mathsf{next\_sibl}^t &= \{(\pi \cdot 1, \pi \cdot 2) \mid \pi \cdot 1 \in \mathsf{nodes}(t)\} \\ \mathsf{label}_c^t &= \{\pi \mid t(\pi) = c\} \end{aligned}$$

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

$$\phi ::= p(x) \mid \mathsf{first\_child}(x,y) \mid \mathsf{next\_sibl}(x,y) \mid \mathsf{label}_c(x) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \forall x.\phi \mid \forall p.\phi$$

A variable assignment α into a tree t maps first-order variables to nodes of t and second-order variables to sets of nodes of 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. Formulas ϕ with n free first-order variables $x_1, ..., x_n$ define n-ary queries, which satisfy for all $t \in T_\Sigma$:

$$\mathsf{query}_{\phi(x_1,\ldots,x_n)}(t) = \{(\alpha(x_1), ..., \alpha(x_n)) \mid t, \alpha \models \phi\}$$

Definition 2. *An n-ary query is* MSO definable *if it is equal to some* $\mathsf{query}_{\phi(x_1,\ldots,x_n)}$.

An equivalent way of defining n-ary queries in MSO is by formulas ϕ with n free second-order variables $p_1, ..., p_n$. For all $t \in T_\Sigma$ let:

$$\mathsf{query}_{\phi(p_1,\ldots,p_n)}(t) = \cup_{t,\alpha\models\phi}\ \alpha(p_1) \times \ldots \times \alpha(p_n)$$

Lemma 1. *An n-ary query is MSO definable iff it is equal to some* $\mathsf{query}_{\phi(p_1,\ldots,p_n)}$.

Proof. We define such queries in MSO by the followig formula $\phi'(x_1, \ldots, x_n)$:

$$\exists p_1 \ldots \exists p_n.\ (\phi(p_1, \ldots, p_n) \wedge p_1(x_1) \wedge \ldots \wedge p_n(x_n))$$

For the converse, we start with some formula $\phi'(x_1, \ldots, x_n)$ and define its query equivalently by some formula $\phi(p_1, \ldots, p_n)$. We use auxilary formulas $p = \{x\}$ defined by $p(x) \wedge \forall y\ (p(y) \rightarrow x{=}y)$. The equality $x{=}y$ used there is a shortcut for $\forall p\ (p(x) \leftrightarrow p(y))$. Now, we define the formula ϕ to be:

$$\exists x_1 \ldots \exists x_n.\ (\phi'(x_1, \ldots, x_n) \wedge p_1 = \{x_1\} \wedge \ldots \wedge p_n = \{x_n\})$$

2.3 Tree automata

We recall the definitions of tree automata and recognizable tree languages [7]. A *tree automaton* A for trees over a binary signature Σ consists of a finite set $\mathsf{states}(A)$, a finite set $\mathsf{rules}(A)$, and a set $\mathsf{final}(A) \subseteq \mathsf{states}(A)$. The rules of A may have two forms:

$$a \rightarrow p \quad \text{or} \quad f(p_1, p_2) \rightarrow p$$

where $f \in \Sigma$ is a binary function symbol, $a \in \Sigma$ a constant and $p, p_1, p_2 \in \mathsf{states}(A)$.

A *run* r of a tree automaton A on a tree t is a function $r : \mathsf{nodes}(t) \rightarrow \mathsf{states}(A)$ that associates states to nodes of t according to the rules of A. Equivalently, we can see runs r of automata A on trees t as tree labeled in $\mathsf{states}(A)$ such that $\mathsf{nodes}(r) = \mathsf{nodes}(t)$. We note $\mathsf{runs}_A(t)$ the set of all runs of A on t. A run r of a tree automaton A on a tree t is called *successful* if it labels the root of t by some state in $\mathsf{final}(A)$.

$$\mathsf{succ\_runs}_A(t) = \{r \in \mathsf{runs}_A(t) \mid r(\epsilon) \in \mathsf{final}(A)\}$$

A tree t is *accepted* by a tree automaton A if A has a successful run on t. The *language* $L(A)$ of tree recognized by a automaton A is the set of trees t that A accepts. A tree language is *regular* if it is recognized by some tree automaton.

A tree automaton A is (bottom-up) *deterministic* if no two of its rules have the same left hand side. It is *unambiguous*, if no tree $t \in T_\Sigma$ permits more than one successful run in $\mathsf{succ\_runs}_A(t)$. Deterministic tree automata are unambiguous since they permit at most one run per tree, while unambiguous automata may be nondeterminstic. They can permit multiple runs on the same tree, of which at most one is successful.

2.4 Queries as tree languages

Another natural way to look at queries is to identify them by tree languages. This idea is similar to Thatcher and Wright's idea to consider models of MSO formulas as tree languages.

Let $\mathbb{B} = \{0, 1\}$ be the set of Booleans. A Boolean tree β is binary tree in which all nodes may be labeled by Booleans, i.e., a tree over the signature $\mathbb{B}$ where Booleans are overloaded to serve both, as binary function symbols and as constants.

It is convenient to define products of trees with the same nodes. More generally, we define products of functions with the same domain. The product of m functions $g_i : C \to D_i$ is the function $g_1 * \ldots * g_m : C \to D_1 \times \ldots \times D_m$ that satisfies for all $c \in C$:

$$(g_1 * \ldots * g_m)(c) = (g_1(c), \ldots, g_m(c))$$

The product $t_1 * \ldots * t_m$ of m trees with the same domain (but possibly different signatures) is the tree whose labeling function is the product of labeling functions of t_1, ..., t_n. A language L of trees over $\Sigma \times \mathbb{B}^n$ corresponds to the following n-ary query:

$$\begin{aligned}\mathsf{query}_L(t) = \{(\pi_1, \ldots, \pi_n) \mid \exists\beta_1, \ldots, \beta_n,\ & t * \beta_1 * \ldots * \beta_n \in L, \\ & \beta_1(\pi_1) = \ldots = \beta_n(\pi_n) = 1\}\end{aligned}$$

Such languages identify queries uniquely, but conversely, the same query may be identified by many different languages.

Definition 3. *An n-ary query in trees over Σ is* regular *iff it is equal to $\mathsf{query}_{L(A)}$ for some tree automaton A over $\Sigma \times \mathbb{B}^n$.*

Theorem 1. *(Thatcher and Wright [23]). An n-ary query in trees is MSO definable iff it is regular.*

MSO formulas $\phi(p_1, \ldots, p_n)$ define languages of trees over $\Sigma \times \mathbb{B}^n$ representing the query $\mathsf{query}_{\phi(p_1,\ldots,p_n)}$. Different formulas may define different languages for the same query. Which formula or language to choose will turn out crucial for what follows.

Given a set S with subset $S' \in S$, we define a characteristic function $\mathsf{c}_{S'} : S \to \mathbb{B}$ so that $\mathsf{c}_{S'}(s) \leftrightarrow s \in S'$ for all $s \in S$. Every subset $P \subseteq \mathsf{nodes}(t)$ defines a characteristic function c_P that we identified with the Boolean trees whose labeling function is c_P. This tree has the same nodes as t. Formulas $\phi(p_1, \ldots, p_n)$ define a tree language over $\Sigma \times \mathbb{B}^n$:

$$L_{\phi(p_1,\ldots,p_n)} = \{t * \mathsf{c}_{\alpha(p_1)} * \ldots * \mathsf{c}_{\alpha(p_n)} \mid t, \alpha \models \phi(p_1, \ldots, p_n)\}$$

Lemma 2. *$\mathsf{query}_{L_{\phi(p_1,\ldots,p_n)}} = \mathsf{query}_{\phi(p_1,\ldots,p_n)}$.*

Similarly, we define $L_{\phi(x_1,\ldots,x_n)}$ by considering all first-order variables x_i as singleton valued second-order variables. Trees $t * \beta_1 * \ldots * \beta_n \in L_{\phi(x_1,\ldots,x_n)}$ are *canonical* in that each of them identifies precisely one tuple of $\mathsf{query}_{\phi(x_1,\ldots,x_n)}(t)$, i.e., all sets $\beta_i^{-1}(1)$ are singletons for $1 \leq i \leq n$.

3 Run-based queries

We now introduce new representation formalisms for MSO-definable n-ary queries, that are directly based on successful runs of tree automata, without signature extension. Our definitions remix ideas from [16, 9, 2] into a new general framework.

3.1 Existential run-based queries

Tree automata can not only accept trees but directly select nodes in successful runs. The idea is that states of tree automata are properties of nodes, which are verified by successful runs.

An *existential run-based* n-ary query $\mathsf{query}^{\exists}_{A,S}$ in binary trees over Σ is given by a tree automaton A over Σ and a set $S \subseteq \mathsf{states}(A)^n$ of so called *selection tuples*. It selects all those tuples of nodes $(\pi_1, \ldots, \pi_n)$ in a tree t that are assigned to a selection tuple by some successful run of A on t:

$$\mathsf{query}^{\exists}_{A,S}(t) = \{(\pi_1, \ldots, \pi_n) \mid \exists r \in \mathsf{succ\_runs}_A(t), (r(\pi_1), \ldots, r(\pi_n)) \in S\}$$

Run-based existential n-ary queries where first proposed by Neven and Bussche [16] in the framework of attribute grammars (these can be seen as tree automata whose states are vectors of attribute values). Their BAG's can express all regular monadic queries, whereas their RAG's are more expressive than regular n-ary queries. Run-based existential n-ary queries that we conjecture to capture MSO where first proposed by Seidl and Berlea [2], in the framework of pushdown forest automata for unranked trees. However, no relation between run-based n-ary queries and MSO-definable queries was established there.

Example 1 *Consider the automaton A_1 over signature $\Sigma = \{f, a\}$. Two runs of A_1 on the tree $f(f(a, a), a)$ are presented in Figure 1. Successful runs of A_1 label the left most a-leaf by 1 and all others by $*$. The ancestors of the left most a-leaf will be assigned to y. All other inner nodes will be marked by $*$. The final states are y and 1. In summary, the automaton A_1 has the following states and rules:*

$$\begin{array}{rlll} \mathsf{states}(A_1) = \{1, *, y\} & a \to 1 & f(1, *) \to y & f(y, *) \to y \\ \mathsf{final}(A_1) = \{1, y\} & a \to * & f(*, *) \to * & \end{array}$$

This automaton is (bottom-up) non-deterministic. It is unambiguous, however, in that no tree permits more than one successful run by A_1. The monadic query $\mathsf{query}^{\exists}_{A_1,\{1\}}$ selects the left most a-leaf. This monadic query cannot be represented by any deterministic tree automaton. Non-determinism is needed to distinguish different occurrences of a-leaves, those that have to be selected from the others. States that are guessed for a node are correct, if they lead to a successful run.

Example 1 illustrates that deterministic tree automata are not sufficient to define all regular monadic queries (even though they can recognize all regular languages). Nevertheless, all regular monadic queries can be expressed with a limited amout of non-determinism (in contrast to regular n-ary queries as we will see). Unambiguous

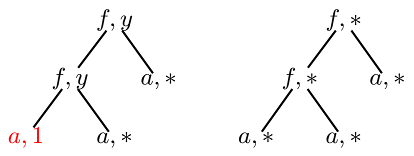

Fig. 1. Selecting the left most a-leaf:$\mathsf{query}^{\exists}_{A_1,\{1\}}$. Only the left run of A_1 is successful.

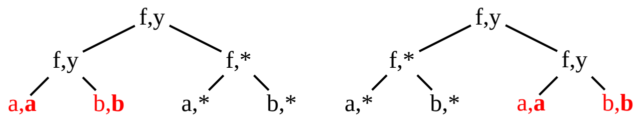

Fig. 2. Selecting pairs of a-leaves and next-sibling b-leaves: $\mathsf{query}^{\exists}_{A_2,\{(\mathbf{a},\mathbf{b})\}}$

automata are enough, as in the example. This means that there always exists at most one correct way of guessing states. This result is well known in the monadic case [16, 3].

Example 2 *Let us define the binary query that selects pairs of a-leaves and next-sibling b-leaves. We assume that the signature is $\Sigma = \{f, a, b\}$ and define the automaton A_2 with $\mathsf{states}(A_2) = \{\boldsymbol{a}, \boldsymbol{b}, *, y\}$ that will produce successful runs of the form of Figure 2. The required tuples are selected by $\mathsf{query}^{\exists}_{A_2,\{(\boldsymbol{a},\boldsymbol{b})\}}$. The automaton A_2 will assign state* ***a*** *to selected a-leaves and state* ***b*** *to the corresponding next-sibling b-leaves. The final state y will be assigned to all common ancestors of the selected pair of leaves: $\mathsf{final}(A_2) = \{y\}$. State $*$ can be assigned to all other nodes. The following rules verify these properties:*

$$\begin{array}{llll} a \rightarrow \boldsymbol{a} & b \rightarrow \boldsymbol{b} & f(*,*) \rightarrow * & f(\boldsymbol{a},\boldsymbol{b}) \rightarrow y \\ a \rightarrow * & b \rightarrow * & f(y,*) \rightarrow y & f(*,y) \rightarrow y \end{array}$$

Every successful run of the automaton A_2 will select a single pair of nodes. Different pairs are separated by different runs so that they cannot be mixed up.

This example illustrates the trick in our representation of n-ary queries. In order to not mix up the components of selected pairs, we separate them in different runs. This trick gives hope that run-based existential n-ary queries can represent all regular n-ary queries. However, it raises doubts on the querying power on unambiguous tree automata in existential run-based n-ary queries.

Our next goal is to show that existential run-based queries capture the class of regular n-ary queries.

Theorem 2. *Existential run-based n-ary queries capture precisely the class of regular n-ary queries.*

Proof. Let us first show that existential run-based n-ary queries are regular. Due to Thatcher and Wright's theorem, it is sufficient to show that existential run-based queries are MSO definable. First note that they are finite unions of existential run-based queries with singleton selection sets:

$$\mathsf{query}^{\exists}_{A,S} = \cup_{(p_1,\dots,p_n)\in S}\ \mathsf{query}^{\exists}_{A,\{(p_1,\dots,p_n)\}}$$

Since MSO definable queries are closed under union it remains to prove that the query $\mathsf{query}^{\exists}_{A,\{(p_1,\dots,p_n)\}}$ with single selection tuple is MSO definable. This can be done by the following formula $\phi(p_1, \dots, p_n)$. Suppose that $\{p_{n+1}, \dots, p_m\} = \mathsf{states}(A) \setminus \{p_1, \dots, p_n\}$:

$$\begin{array}{lll}
\exists p_{n+1} \dots \exists p_m. & \forall x.\ \overset{+}{\vee}_{p\in\mathsf{states}(A)}\ p(x) & \\
\quad \wedge\forall x.\ \mathsf{root}(x) \to \vee_{p\in\mathsf{final}(A)}\ p(x) & & \\
\quad \wedge\forall x.\ p(x) \to \vee_{a\to p\in\mathsf{rules}(A)}\ \mathsf{label}_a(x)\vee & & \\
& \quad \vee_{f(p_1',p_2')\to p\in\mathsf{rules}(A)}\ \mathsf{label}_f(x)\wedge & \\
& & \exists x_1.\ \mathsf{first\_child}(x,x_1) \wedge p_1'(x_1)\wedge \\
& & \exists x_2.\ \mathsf{next\_sibl}(x_1,x_2) \wedge p_2'(x_2)
\end{array}$$

The formula means that every node of the tree must be assigned to a unique state, that the root node must be assigned to a final state, and that every state must be justified, either by a rule labeling leaves or a rule labeling inner nodes.

Let us prove now that every regular query is equal to some existential run-based query. Let $\mathsf{query}_{L(A)}$ be a regular n-ary query for some tree automaton A over $\Sigma \times \mathbb{B}^n$. We compute an automaton $\mathsf{proj}(A)$ over Σ by projecting Booleans from the labels into states. Let $\mathsf{states}(\mathsf{proj}(A)) = \mathsf{states}(A)\times\mathbb{B}^n$, $\mathsf{final}(\mathsf{proj}(A)) = \mathsf{final}(A)\times\mathbb{B}^n$. The rules of $\mathsf{proj}(A)$ are generated by the following schema for all $a, f \in \Sigma$, $p_1, p_2, p \in \mathsf{states}(A)$ and $b, b_i, b_i^1, b_i^2 \in \mathbb{B}$ where $1 \le i \le n$:

$$\frac{(a, b_1, ..., b_n)\to p \in \mathsf{rules}(A)}{a\to(p, b_1, ..., b_n) \in \mathsf{rules}(\mathsf{proj}(A))}$$

$$\frac{(f, b_1, ..., b_n)(p_1, p_2)\to q \in \mathsf{rules}(A)}{f((p_1, b_1^1, ...b_1^n), (p_2, b_2^1, ..., b_2^n))\to(p, b_1, ..., b_n) \in \mathsf{rules}(\mathsf{proj}(A))}$$

We define the selection set $S \subseteq \mathsf{states}(\mathsf{proj}(A))^n$ by $S = Q_1 \times ... \times Q_n$ such that for all $1 \le i \le n$: $Q_i = \{(q, b_1, ..., b_n) \in \mathsf{states}(\mathsf{proj}(A)) \mid b_i = 1\}$. We show that:

$$\mathsf{query}_{L(A)} = \mathsf{query}^{\exists}_{\mathsf{proj}(A),S}$$

This follows from that for any term $t * \boldsymbol{\beta}$ over $\Sigma \times \mathbb{B}^n$: $\mathsf{runs}_{\mathsf{proj}(A)}(t) = \{r * \boldsymbol{\beta} \mid r \in \mathsf{runs}_A(t * \boldsymbol{\beta})\}$ and $\mathsf{succ\_runs}_{\mathsf{proj}(A)}(t) = \{r * \boldsymbol{\beta} \mid r \in \mathsf{succ\_runs}_A(t * \boldsymbol{\beta})\}$.

3.2 Universal run-based queries

Universal run-based query quantify universally rather than existentially over successful runs. Universal n-ary queries were first introduced by Neven and Bussche [16] in the

$$\begin{array}{lll}
\mathsf{rules}(A_3): & a \to 0 & f(1,2) \to y \\
 & a \to 1 & f(0,y) \to y \\
 & a \to 2 & f(y,0) \to y \\
 & & f(0,0) \to 0 \\
 & & f(0,0) \to 1 \\
 & & f(0,0) \to 2 \\
\text{final states:} & \multicolumn{2}{l}{\mathsf{final}(A_3) = \{y,1,2\}} \\
\text{selection tuples:} & \multicolumn{2}{l}{S_3 = \{(1,2),(0,0),(y,0),(0,y)\}}
\end{array}$$

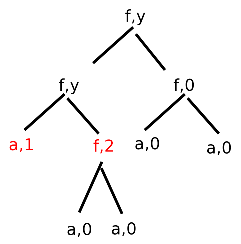

Fig. 3. An example for a universal run-based query: $\mathsf{next\_sibl} = \mathsf{query}^{\forall}_{A_3,S_3}$. The presented successful run refutes wrong partners for the left-most a-leaf and its next sibling.

framework of attribute grammars (universal BAGs and RAGs). In the monadic case, they were reproposed in the selection automata of Frick, Grohe, and Koch [9].

$$\mathsf{query}^{\forall}_{A,S}(t) = \{(\pi_1,\ldots,\pi_n) \ \mid \forall r \in \mathsf{succ\_runs}_A(t), (r(\pi_1),\ldots,r(\pi_n)) \in S\}$$

An example is given in Figure 3. We represent the binary query $\mathsf{next\_sibl}$ universally. Successful runs of automaton A_3 will assign the state pair $(1,2)$ to at most one node pair satisfying the query. Descendants and cousins of these nodes will be assigned to state 0, all others (ancestors in fact) to y. The required query can be expressed existentially by $\mathsf{query}^{\exists}_{A_3,\{(1,2)\}}$.

Runs in universal queries refute all those tuples that they don't select. Thus, one needs sufficiently many selection states so that correct tuples are never rejected. In the example, selected pairs will always be labeled in $S_3 = \{(1,2),(0,0),(0,y),(y,0)\}$. All other node pairs can be refuted by successful runs that assign state pairs in the complement of S_3. Hence $\mathsf{query}^{\exists}_{A_3,\{(1,2)\}} = \mathsf{query}^{\forall}_{A_3,S_3}$.

Theorem 3. *Existential and universal queries have the same expressiveness.*

This theorem has been shown previously but only in the monadic case [16]. The proof there relies on the two phase querying answering algorithm, which fails for non-monadic queries. The above theorem, however, remains true for n-ary queries. This follows from Thatcher and Wright's theorem and our Theorem 2.

Proof. We define the complement q^c of a query q such that for all trees $t \in T_\Sigma$:

$$q^c(t) = \mathsf{nodes}(t)^n \setminus q(t)$$

Existential queries are regular and thus MSO-definable, so their complements are MSO-definable, thus regular, and thus definable by existential run-based queries, too (Theorems 1 and 2).

Furthermore, the definitions of existential and universal queries are dual modulo complementation, i.e., for every tree automaton A with selection tuples $S \subseteq \mathsf{states}(A)^n$:

$$\mathsf{query}^{\forall}_{A,S} = (\mathsf{query}^{\exists}_{A,\mathsf{states}(A)\setminus S})^c$$

Since complements of existential queries are existential, it follows that universal queries are existential too. Vice versa, let q be an existential query. So q^c is equal to $\mathsf{query}^{\exists}_{A,S}$

for some A, S. Hence, $q = \mathsf{query}^{\forall}_{A,\mathsf{states}(A)\setminus S}$, i.e., q is can be represented by a universal query.

Proposition 1. *The combined complexities of computing existential and universal run-based queries are both polynomial.*

Proof. We can compute $\mathsf{query}^{\exists}_{A,\{(p_1,\ldots,p_n)\}}(t)$ naivly by enumerating all n-tuples of nodes $(\pi_1, \ldots, \pi_n)$ in t, and check whether A permits a successful run r such that $r(\pi_i) = p_i$ for all $1 \leq i \leq n$. This can be done in time $O(|A| * |t|^{n+1})$. For computing $\mathsf{query}^{\forall}_{A,\{(p_1,\ldots,p_n)\}}(t)$ it is sufficient to compute and complement $\mathsf{query}^{\exists}_{A,\mathsf{states}(A)\setminus S}(t)$.

4 Unambiguous tree automata

Our next goal is to investigate the querying power of unambiguous tree automata in the n-ary case. These provide a limited form of nondeterminism that is sufficient to capture all regular monadic queries – in contrast to deterministic tree automata [16, 3].

Beside of their general interest, run-based queries by unambiguous tree automata are the query representation formalism underlying a recent approach to query induction for Web information extraction [5] by methods of grammatical inference [19]. The querying power of unambiguous tree automata characterizes the coverage of such approach.

4.1 Finite unions of Cartesian closed queries

We call a n-ary query *Cartesian closed* if it is a Cartesian product of monadic queries, and *unambiguous* if it has the form $\mathsf{query}^{\exists}_{A,S}$ for some unambiguous tree automaton A.

Theorem 4. *Run-based queries by unambiguous tree automata capture the class of finite unions of Cartesian closed regular queries.*

Proof. Every $\mathsf{query}_{A,S}$ is a finite union $\cup_{s\in S}\mathsf{query}_{A,\{s\}}$ with singleton selection sets. If A is unambiguous then we can represent queries with singleton selection sets as by the following Cartesian product:

$$\mathsf{query}_{A,\{(p_1,\ldots,p_n)\}} = \mathsf{query}_{A,\{p_1\}} \times \ldots \times \mathsf{query}_{A,\{p_n\}}$$

This is, since all components of a tuple will be selected in the same successful run.

To prove the converse, let us first note that a cartesian closed regular query is unambiguous. Indeed regular monadic queries are known to be unambiguous run-based [16, 3]. Cartesian products of unambiguous queries are clearly unambiguous too.

It remains to prove that finite unions of unambiguous queries are unambiguous. Let $q = \bigcup_{j=1}^{k} \mathsf{query}^{\exists}_{A_i,S_i}$ be such a union. Let us first assume that all A_i are strictly unambiguous in that they permit precisely one successful run per tree. We then define an unambiguous automaton A as the product of the A_i's such that $\mathsf{final}(A) = \mathsf{final}(A_1) \times \ldots \times \mathsf{final}(A_k)$. Let $\mathsf{proj}_i(p)$ be the $i-$th component of a state p of A. We let the selection set S to be the set of all tuples $(p_1, \ldots, p_n) \in \mathsf{states}(A)^n$ for which there exists $i \in \{1, \ldots, k\}$ such that $(\mathsf{proj}_i(p_1), \ldots, \mathsf{proj}_i(p_n)) \in S_i$. Thus, $q = \mathsf{query}^{\exists}_{A,S}$.

Second note that we can make all unambiguous tree automata A_i strictly unambiguous. Let $\bar{A}_i$ the deterministic automaton accepting the trees not accepted by A_i; assuming A_i and $\bar{A}_i$ have disjoint sets of states, we define A'_i as $A_i \cup \bar{A}_i$. This automaton A'_i is strictly unambiguous and moreover, $\mathsf{query}^{\exists}_{A'_i,S_i} = \mathsf{query}^{\exists}_{A_i,S_i}$.

The proof shows that unambiguous queries are closed under union, and clearly, they are closed under intersection. So if $\mathsf{query}_{\phi(x_1,\ldots,x_n)}$ and $\mathsf{query}_{\phi'(x_1,\ldots,x_n)}$ are unambiguous MSO queries, so are $\mathsf{query}_{\phi\wedge\phi'(x_1,\ldots,x_n)}$ and $\mathsf{query}_{\phi\vee\phi'(x_1,\ldots,x_n)}$. In other words, the class of MSO formulas defining unambiguous queries is closed under conjunction and disjunction.

Proposition 2. *A query is unambiguous iff it can be expressed by a disjunction of conjunctions of MSO formulas with a single free variable each.*

Proof. This follows from Theorem 4. Consider a Cartesian product $\mathsf{query}_{A,\{p_1\}} \times \ldots \times \mathsf{query}_{A,\{p_n\}}$. Suppose that $\mathsf{query}_{A,\{p_i\}}$ is equal to $\mathsf{query}_{\phi_i(x_i)}$, then their Cartesian product can be expressed by $\phi_1 \wedge \ldots \wedge \phi_n(x_1, \ldots, x_n)$. Unions of such queries can be expressed by disjunctions. The converse follows from the closure properties discussed above.

Corollary 1. *The class of unambiguous run-based queries is closed under complementation (and thus all Boolean opertions).*

Proof. On the logical side, complementation corresponds to negation. By computing disjunctive normal forms, negations can be pushed down into monadic MSO formulas.

4.2 Faithful language representations of unambiguous queries.

To get a decidable characterization of unambiguity, we will exploit properties of some tree language representations of a query.

Definition 4. *Let L be a language of trees over $\Sigma \times \mathbb{B}^n$.*
*L is said $k-$*faithful *if* $\sup_{t\in T_\Sigma} |\{t * \beta \mid t * \beta \in L\}| \leq k$.
L is said faithful *if it is $k-$faithful for some k.*

Proposition 3. *A language L is faithful if and only if it is a finite union of $1-$faithful languages. Furthermore, a regular language is faithful if and only if it is a finite union of regular $1-$faithful languages.*

Proof. The first part of the property is straightforward from the definition. For the second one, it remains to prove that a regular language is faithful only if it is a finite union of regular $1-$faithful languages. More precisely, we will decompose a $k-$faithful language in k regular $1-$faithful languages. Let $<$ be the lexicographic ordering on $\mathbb{B}^n$ and let us define a total strict ordering $\prec$ on trees from $T_{\Sigma\times\mathbb{B}^n}$ as follows: $t * \beta \prec t * \beta'$ if (i) $\beta(\epsilon) < \beta'(\epsilon)$, or (ii) $\beta(\epsilon) = \beta'(\epsilon)$ and $t_1 * \beta_1 \prec t_1 * \beta'_1$ or (iii) $\beta(\epsilon) = \beta'(\epsilon)$, $t_1 * \beta_1 = t_1 * \beta'_1$ and $t_2 * \beta_2 \prec t_2 * \beta'_2$ (where $t_j * \beta_j$ is the subtree at position j of $t * \beta$). Obviously, $\prec$ is a recognizable relation of $T_{\Sigma\times\mathbb{B}^n} \times T_{\Sigma\times\mathbb{B}^n}$.

So, L_1, the set of greatest elements in the sense of $\prec$ from L is effectively recognizable. By induction, L_i, the set of i-th greatest elements in the sense of $\prec$ from L is recognizable too. As L is $k-$faithful, L is $\bigcup_{i=1}^{n} L_i$.

Proposition 4. *1. An n-ary query is Cartesian closed query iff it has the form query_L for some $1-$faithful tree language L.*

2. An n-ary query is a finite union of Cartesian closed queries iff it has the form query_L for some faithful tree language L.

3. A regular n-ary query is unambiguous iff it has the form query_L for some faithful recognizable tree language L.

Proof. A query q that is Cartesian closed, for all $t \in T_\Sigma$, can be written as $E_1^t \times \ldots \times E_n^t$. For each $t \in T_\Sigma$, we consider the tree $t * \beta^q$ defined by: for all $\pi \in \mathsf{nodes}(t)$ and $\beta^q(\pi) = (b_1, \ldots, b_n)$, $b_i \leftrightarrow (\pi \in E_i^t)$. Defining L_q as $\{t * \beta^q \mid t \in T_\Sigma\}$, it is easy to verify that $q = \mathsf{query}_{L_q}$ and is $1-$faithful. Conversely, if q is represented by a $1-$faithful language, q is clearly Cartesian closed. The rest of the proposition is then directly obtained by noticing that $\bigcup_{i=1}^n \mathsf{query}_{L_i} = \mathsf{query}_{\bigcup_{i=1}^n L_i}$ and by using the Proposition 3 and Theorem 4.

4.3 Deciding unambiguity of queries

We show in this section that one can decide whether a regular n-ary query is unambiguous, or equivalently by Theorem 4 whether the query is a finite union of Cartesian closed regular queries.

Note that deciding whether a regular query is Cartesian closed is straightforward using relationship between regular queries and MSO formulas and using that the Cartesian closed property is MSO-definable. Considering finite unions of Cartesian closed regular queries requires more sophisticated techniques.

Let q a query. We will define, $\mathsf{max}(q)$, a tree language that represents q and has good compactness properties: $\mathsf{max}(q)$ will be faithful as soon as q has a faithful representation. Roughly speaking, a labeled tree $t * \beta$ will be in $\mathsf{max}(q)$ if it is correct -the nodes selected by β are in $q(t)$- and maximal -no 1 can be added while keeping correct-.

Let us define precisely $\mathsf{max}(q)$ for q a (regular) query defined by the (MSO) formula $\phi_q(x_1, \ldots, x_n)$. Let $\phi_q^{\mathsf{max}}(p_1, \ldots, p_n)$ be the following formula:

$$\forall x_1 \ldots \forall x_n \; p_1(x_1) \wedge \ldots \wedge p_n(x_n) \rightarrow \phi_q(x_1, \ldots, x_n)$$
$$\wedge_i \forall x_i \; \neg p_i(x_i) \rightarrow \exists x_1 \ldots \exists x_{i-1} \exists x_{i+1} \ldots \exists x_n \wedge_{j \neq i} p_j(x_j) \wedge \neg\phi_q(x_1, \ldots, x_n)$$

Then, we define $\mathsf{max}(q)$ as $L_{\phi_q^{\mathsf{max}(q)}(p_1, \ldots, p_n)}$: for a regular query q, $\mathsf{max}(q)$ is recognizable (and we can effectively construct an automaton for it from an automaton or an MSO formula defining q).

Lemma 3. *A query q is a finite union of Cartesian closed queries iff $\mathsf{max}(q)$ is faithful.*

Proof. By Proposition 4 we just have to prove that if the query q is a finite union of Cartesian closed queries, $\mathsf{max}(q)$ is faithful. Let q a finite union of Cartesian closed queries. There exists some natural number k s.t. $q = \bigcup_{j=1}^k q_j^1 \times \ldots \times q_j^n$, each q_j^i being a monadic regular query.

Let t be a tree from T_Σ. For each $1 \leq i \leq n$, we define $\equiv_i$, an equivalence relation on $\mathsf{nodes}(t)$ by $\pi \equiv_i \pi'$ if for all $(\pi_1, ..., \pi_{i-1}, \pi_{i+1}, ..., \pi_n)$, $(\pi_1, ..., \pi_{i-1}, \pi, \pi_{i+1}, ..., \pi_n)$

belongs to $q(t)$ iff $(\pi_1, ..., \pi_{i-1}, \pi', \pi_{i+1}, ..., \pi_n)$ belongs to $q(t)$. This just means that π and π' are, in some sense, interchangeable in i-th position. Then, let π and π' be two nodes. If for each $1 \leq j \leq k$, π belongs to $q^i_j(t)$ iff π' belongs to $q^i_j(t)$, then $\pi \equiv_i \pi'$. This implies that $\equiv_i$ is of finite index bounded by 2^k. Now let $t * \beta$ be a term in $\mathsf{max}(q)$. Let π selected in the i-th position by β, i.e., such that $\beta(\pi)_i = 1$. Then, by maximality of $t * \beta$, for each π' s.t. $\pi \equiv_i \pi'$, we have also $\beta(\pi')_i = 1$. This implies that $\{\pi \mid \beta(\pi)_i = 1\}$ is an union of equivalence classes for $\equiv_i$. So, the cardinality of the set $\{t * \beta \mid t * \beta \in \mathsf{max}(q)\}$ is upper-bounded by $2^{n.2^k}$.

Let us note that if $\mathsf{max}(q)$ is faithful as soon there is a faithful representation of q, it is non necessarly the "most faithful" one or the "less redundant" one. Indeed let us suppose that $q(t) = \{(n_1, n_1), (n_1, n_2), (n_1, n_3), (n_2, n_1), (n_2, n_4)\}$, for some t. A $2-$faithfull representation can be based on $\{n_1\} \times \{n_1, n_2, n_3\}, \{n_2\} \times \{n_1, n_4\}$ whereas in $\mathsf{max}(q)$, a tree associated with $\{n_1, n_2\} \times \{n_1\}$ will be added.

Now, let q a regular query (given by a tree automaton or a formula): first we construct A a deterministic automaton recognizing $\mathsf{max}(q)$. Then, we compute an automaton $\mathsf{proj}(A)$ as in Theorem 2. Clearly the numbers of accepting runs on t in $\mathsf{proj}(A)$ is the cardinal of $\{t * \beta \mid t * \beta \in \mathsf{max}(q)\}$.

A tree automaton A is said k-ambiguous if for any tree $t \in T_\Sigma$, there exists at most k accepting runs for t in A. The degree of ambiguity of an automaton A is bounded if A is k-ambiguous for some natural number k.

So, by what precedes, q is unambiguous iff the degree of ambiguity of $\mathsf{proj}(A)$ is bounded, which can be decided:

Theorem 5 (Seidl [21]). *Whether the degree of ambiguity of a tree automaton is bounded is decidable. Furthermore their degree of ambiguity can be computed.*

As all contructions are effective, it gives us a decision procedure for unambiguousness of q. Furthermore, let us note that if this gives us a way to compute an unambiguous automaton computing q. Indeed, let us suppose that the degree of A is k. You can build an automaton A_k simulating A on trees which have at least k accepting runs in A -by making the product of k copies of A and checking the k runs are different-; as the degree of A is k, A_k will be unambiguous. Then, you can build an unambiguous automaton A_{k-1} simulating A on trees which have exactly $k-1$ accepting runs in A, by a similar construction and checking that the tree is not accepted by A_k. By iterating the construction, you can build $(A_i, S_i)_{i=1}^k$, with A_i unambiguous automata simulating A on trees which have exactly i accepting runs in A: q is the union of the corresponding queries and by using effective closure under union, you can then build an unambiguous automaton for q.

Theorem 6. *Ambiguity of a query is decidable. Furthermore, when a query q is unambiguous, (A, S) with A an unambiguous automaton s.t. $q = \mathsf{query}^{\exists}_{A,S}$ can effectively be constructed.*

To end this section, let us say a few words about the monadic case. In this case, if A is a deterministic automaton recognizing $\mathsf{max}(q)$, let us note that the automaton $\mathsf{proj}(A)$ is always unambiguous -monadic queries are known unambiguous-. Furthermore, it can be easily checked that this automaton is deterministic -after deleting the unproductive states- iff the query is determinsitic. So, it provides a way to decide determinism and to compute, when possible, a deterministic automaton which computes the query.

5 Querying unranked trees

Our results carry over to unranked trees by hedge automata [4]. An unranked tree is build from a set of constants $a, b \in \Sigma$ by the abstract syntax $t ::= a(t_1, \ldots, t_n)$ where $n \geq 0$. A *hedge automaton* H over Σ consists of a set $\mathsf{states}(H)$, a set $\mathsf{final}(H) \subseteq \mathsf{states}(H)$, and a set $\mathsf{rules}(H)$ of rules of the form $a(A) \to p$ where A is finite word automaton with alphabet $\mathsf{states}(H)$ and $p \in \mathsf{states}(H)$. Runs of hedge automata H on unranked trees t are functions $r : \mathsf{nodes}(t) \to \mathsf{states}(H)$ defined as

$$\frac{\begin{array}{ll} t = a(t_1, \ldots, t_n) & \forall 1 \leq i \leq n : r_i \in \mathsf{runs}_H(t_i) \\ a(A) \to p \in \mathsf{rules}(H) & r_1(\epsilon) \ldots r_n(\epsilon) \in L(A) \end{array}}{p(r_1, \ldots, r_n) \in \mathsf{runs}_H(t)}$$

Queries for the class of unranked trees over Σ are defined as before. The notion of unambiguity (that is the existence of at most one run for a tree) carries over literally to hedge automata (in contrast to determinism). The same holds for the notions of run-based queries by hedge automata.

Theorem 7. *Existential and universal n-ary queries in unranked trees by runs of hedge automata capture* MSO *over run-ranked trees (comprising the* next_sibl*-relation). Run-based queries by unambiguous hedge automata capture the class of finite unions of Cartesian closed queries. This property is decidable. Queries by unambiguous hedge automata have linear combined complexity.*

We only give a sketch of the proof. The main idea is to convert queries by hedge automata into queries by stepwise tree automata [6] for which all results apply. Stepwise tree automata over an unranked signature Σ are tree automata for binary trees with constants in Σ and a single binary function symbol @. Stepwise tree automata can be understood as tree automata that operate on Currified binary encodings of unranked trees. The Currification of $a(b, c(d, e, f), g)$ for instance is the binary tree $a@b@(c@d@e@f)@g$.

Stepwise tree automata were proved to have two nice properties that yield a simple proof of the theorem. 1) N-ary queries by hedge automata can be translated to n-ary queries by stepwise automata in linear time, and conversely in polynomial time. The back and forth translations preserve unambiguity. 2) All presented results on run-based n-ary queries for binary trees apply to stepwise tree automata.

References

1. R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web Information Extraction with Lixto. In *The VLDB Journal*, pages 119–128, 2001.
2. A. Berlea and H. Seidl. Binary queries for document trees. *Nordic Journal of Computing*, 11(1):41–71, 2004.

3. R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *JCSS*, 61(1):1–50, 2000.
4. A. Bruggemann-klein, D. Wood, and M. Murata. Regular tree and regular hedge languages over unranked alphabets: Version 1, Apr. 07 2001.
5. J. Carme, A. Lemay, and J. Niehren. Learning node selecting tree transducer from completely annotated examples. In *7th International Colloquium on Grammatical Inference*, LNAI. Springer Verlag, 2004.
6. J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In *RTA*, volume 3091 of *LNCS*, pages 105 – 118. Springer Verlag, 2004.
7. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997.
8. W. W. W. Consortium. XQuery: the W3C query language for XML – W3C working draft. Available at http://www.w3.org/TR/xquery/, 2001.
9. M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees. In *Proc. LICS 2003*, 2003.
10. G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. In PODS, pages 17–28, 2002.
11. G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proceedings of the* 17^{th} *LICS*, Copenhagen, 2002. IEEE Press.
12. G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto data extraction project - back and forth between theory and practice. In *Proceedings of the symposium on Principles of database systems*. ACM press, 2004.
13. R. Kosala, J. V. den Bussche, M. Bruynooghe, and H. Blockeel. Information extraction in structured documents using tree automata induction. In *Proc. of the 6th International Conference Principles of Data Mining and Knowledge Discovery (PKDD-2002)*, pages 299 – 310, 2002.
14. M. Marx. Conditional XPath, the first order complete XPath dialect. In *Proceedings of the symposium on Principles of database systems*, pages 13–22. ACM press, 2004.
15. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In *Foundations of Software Technology and Theoretical Computer Science*, pages 134–145, 1998.
16. F. Neven and J. V. D. Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, 49(1):56–100, 2002.
17. F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS-00)*, pages 145–156, N. Y., May 15–17 2000. ACM Press.
18. F. Neven and T. Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.
19. J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, pages 49–61, 1992.
20. T. Schwentick. On diving in trees. In *MFCS '00: Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 660–669, London, UK, 2000. Springer-Verlag.
21. H. Seidl. On the finite degree of ambiguity of finite tree automata. In *FCT*, pages 395–404, 1989.
22. J. W. Thatcher and J. B. Wright. Generalized finite automata. *Notices Amer. Math. Soc.*, 820, 1965. Abstract No 65T-649.
23. 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.
24. W3C. XSL Transformations (XSLT), 1999. `http://www.w3.org/TR/xslt`.