N-ary Queries by Tree Automata

Joachim Niehren, Laurent Planque, Jean-Marc Talbot, Sophie Tison

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

We investigate n-ary node selection queries in trees by successful runs of tree automata. We show that run-based n-ary queries capture MSO, contribute algorithms for enumerating answers of n-ary queries, and study the complexity of the problem. We investigate the subclass of run-based n-ary queries by unambiguous tree automata.

Keywords: XML, databases, information extraction, logic, automata, types, pattern.

1 Introduction

Node selection is the most widespread database querying problem in the context of XML. Beside other applications, node selection is basic to XML transformation languages (Query, XSLT, XDuce, CDuce, tree transducer, etc [13, 7, 15]) and of interest for Web information extraction (Lixto, Squirrel, etc [1, 12, 5]).

Monadic node selection queries in trees define sets of nodes, while *n-ary node selection queries* define sets of n-tuples of nodes. Binary queries, for instance, can be used to select all pairs of products and prices in XML or HTML documents created from the database of some company. Monadic queries have attracted most attention so far, in particular those specified in the W3C standard *XPath* that is used by XQuery and XSLT, or similar path based query languages [17]. *Monadic Datalog* yields attractive alternatives for expressing monadic queries, in particular for visual Web information extraction [11]. More general n-ary queries have been promoted by XML programming languages with pattern matching such as XDuce and CDuce [13, 7]. Their *patterns* or *types* with n capture variables specify n-ary node selection queries in trees.

Monadic second-order logic (MSO) is the classical language for defining regular node selection queries in trees [21]. Every formula of MSO with n free node variables specifies an n-ary query. MSO is highly expressive, succinct, and robust under many wishful operations. Its usage, however, remains limited due to its high combined complexity in query answering. *Tree automata* provide an equally expressive alternative, according to Thatcher and Wright's 1968 theorem [21]. They avoid the algorithmic complexity of MSO at the cost of lower succinctness. N-ary queries are seen as languages of trees whose nodes are annotated by bit vectors of length n, which may be recognizable by tree automata or not.

In this paper, we investigate the more recent approach of defining n-ary queries by *successful runs of tree automata* [2, 13, 18, 10, 19]. Successful runs annotate all nodes of a tree by states. Given a *selection set* of n-tuples of states, a successful run selects all those n-tuples of nodes that it annotates in the selection set. We study the two cases of ranked and unranked trees. In the unranked case, essentially the same representation formalism has been proposed previously by Berlea and Seidl [2], called n-ary queries

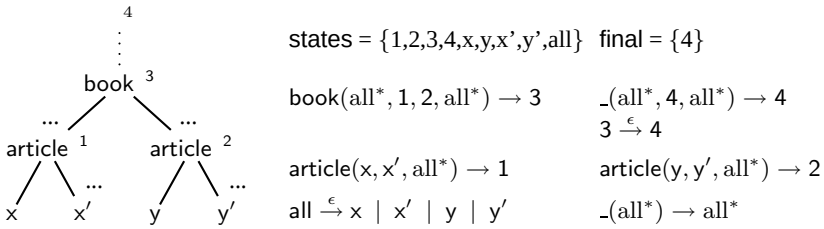

Fig. 1. Pattern as tree automata; matches correspond to successful runs.

by *forest grammars*. In the ranked case, run-based n-ary queries have been proposed by Hosoya and Pierce [13] in terms of *pattern automata*.

N-ary queries by tree pattern are most closely related to run-based queries by tree automata [13]. This is illustrated by the example in Fig. 1. The nodes of the tree pattern on the left become states of the automaton on the right. The root node of the pattern becomes the unique final state. The only selecting n-tuple of automaton states is the n-tuple of capture variables of the pattern. The rules of the automaton express the semantics of the pattern. They can be inferred compositionally. Matches of the pattern correspond to successful runs of the automaton.

In this paper, we prove the folk theorem that run-based n-ary queries capture MSO, to our knowledge for the first time. We then present a deterministic algorithm that can enumerate all answers of an n-ary query by an automaton A with selection set $S \subseteq \mathsf{states}(A)^n$ in time $O(|S| * |A| * |t|^n)$. The combined complexity of run-based n-ary queries is thus in deterministic polynomial time for fixed tuple size n. We also prove that this is not the case if we do not bound the tuple size n.

We then investigate the querying power of *unambiguous tree automata*. Unambiguity limits the amount of nondeterminism to at most one successful run per tree, which is more permissive than imposing bottom-up or top-down determinism. Monadic queries by unambiguous tree automata are of particular interest for query induction [5]. They are known to capture the class of monadic MSO-definable queries (since they are the IBAGs of [18]) in contrast to deterministic tree automata.

For the n-ary case, however, we prove that run-based queries by unambiguous automata 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 can compute represention of all query answers in time $O(n * |S| * |A| * |t|)$. Emptiness is thereby decidable in polynomial time even for unbounded tuple size n. Finally, we show that it is decidable whether an MSO defined query belongs to that restricted class. We reduce this problem to testing the boundedness of the degree of ambiguity of tree automata [20].

2 MSO definable and regular queries

We develop our theory of n-*ary queries for binary trees* which will be sufficient to deal with unranked trees (see Section 6). This section starts with Thatcher and Wright's theorem [21], slightly reformulated in terms of querying rather than recognition.

Let Σ be a finite signature of binary function symbols f and constants a. A *binary tree* $t \in T_\Sigma$ is a ground term over Σ. A *node* π of a tree t is a word in $\{1,2\}^*$ that is the relative address of some subtree starting from the root. 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 node is a *leaf* if it has no child, otherwise it is an *inner node*. 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, such that $\forall t \in T_\Sigma : q(t) \subseteq \mathsf{nodes}(t)^n$.*

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 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 $\pi \cdot 2$. As another example, 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 last query can indeed be expressed by RAG's [18] but cannot be defined in MSO.

In MSO, binary trees $t \in T_\Sigma$ are seen as *logical structures*, whose domain 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.

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

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 $t \in T_\Sigma$:

$$q_{\phi(x_1,...,x_n)}(t) = \{(\alpha(x_1), ..., \alpha(x_n)) \mid t, \alpha \models \phi\}$$

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

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:

$$q_{\phi(p_1,\ldots,p_n)}(t) = \bigcup_{t,\alpha \models \phi} \alpha(p_1) \times \ldots \times \alpha(p_n)$$

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

A *tree automaton* A for binary trees [9] over signature Σ consists of two finite sets $\mathsf{final}(A) \subseteq \mathsf{states}(A)$ and a set $\mathsf{rules}(A)$ with elements of the form $a \to p$ or $f(p_1, p_2) \to p$ 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 mapping $r : \mathsf{nodes}(t) \to \mathsf{states}(A)$ that associates states to nodes of t according to the rules of A. Equivalently, we can see runs as trees labeled in $\mathsf{states}(A)$ such that $\mathsf{nodes}(r) = \mathsf{nodes}(t)$. A run is *successful* if it labels the root of the tree by a final state, *i.e.* if $r(\epsilon) \in \mathsf{final}(A)$. We write $\mathsf{runs}_A(t)$ for the set of all runs of A on t and $\mathsf{succ\_runs}_A(t)$ for the subset of successful runs. A tree t is *accepted* by a tree automaton A if it permits a successful run by A. The *tree language* $L(A)$ recognized by an automaton A is the set of trees t accepted by A. A tree language is *regular* if it is recognized by some tree automaton.

Queries can be viewed as tree languages. This perspective is close to that of Thatcher and Wright, who view models t, α of MSO formulas as trees t annotated by bit vectors encoding α. Sets of models become languages of annotated trees.

Let $\mathbb{B} = \{0, 1\}$ be the set of Booleans. A Boolean tree β is a binary tree whose nodes are labeled by Booleans (here, Booleans serve both as binary function symbols and as constants). As an auxilary notion for formalising compositions of trees with their annotations, we define products of functions with the same domain. The product of m functions $g_i : C \to D_i$ is the function $g_1 * \ldots * g_m : C \to D_1 \times \ldots \times D_m$ such that

$$(g_1 * \ldots * g_m)(c) = (g_1(c), \ldots, g_m(c)) \qquad \text{for all } c \in C$$

Considering trees as functions, 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 $t_1, \ldots, t_n$. A language L of annotated trees over $\Sigma \times \mathbb{B}^n$ corresponds to the following n-ary query:

$$q_L(t) = \{(\pi_1, \ldots, \pi_n) \mid \exists \beta_1, \ldots, \beta_n,\ t * \beta_1 * \ldots * \beta_n \in L,\ \beta_1(\pi_1) = \ldots = \beta_n(\pi_n) = 1\}$$

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

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

Theorem 1. *([21]). 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 $q_{\phi(p_1,\ldots,p_n)}$. Different formulas may define different languages for the same query. Which formula or language to choose to define n-ary queries will turn out to be crucial for what follows.

Given sets $S' \subseteq S$, 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 language of annotated trees over the signature $\Sigma \times \mathbb{B}^n$: $L_{\phi(p_1,\ldots,p_n)} = \{t * \mathsf{c}_{\alpha(p_1)} * \ldots * \mathsf{c}_{\alpha(p_n)} \mid t, \alpha \models \phi(p_1, \ldots, p_n)\}$.

Lemma 2. *An MSO-formula and the language of annotated trees encoding its models define the same query:* $q_{\phi(p_1,\ldots,p_n)} = q_{L_{\phi(p_1,\ldots,p_n)}}$.

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

3 Run-based queries

Boolean annotations of trees are not necessary to define queries by trees automata. Alternatively, one can use successful runs of tree automata to annotate trees by states, and then select from these state annotations. The idea is that automata states are properties of nodes, which can be verified for nodes by successful runs.

An *existential run-based* n-ary query $\mathsf{q}^{\exists}_{A,S}$ 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:

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

Existential run-based n-ary queries were proposed by Neven and Van den Bussche [18] in the framework of attribute grammars (these can be seen as tree automata whose states are vectors of attribute values). Their BAG's correspond to our monadic case, while their RAG's are more expressive than our n-ary case. Existential run-based n-ary queries in binary trees (with a first match semantics) were proposed by Hosoya and Pierce [13][1]. Seidl and Berlea [2] define run-based n-ary queries for unranked trees (by forest grammars), and present an query answering algorithm for the binary case.

It is known from [18] that monadic existential run-based queries capture the class of monadic MSO definable queries. The analogous result for n-ary existential run-based queries might be expected. It holds indeed as we will prove in Theorem 2.

An example is given in Fig. 2. We consider the binary query that selects pairs of a-leave and next-sibling b-leaves, over the signature $\Sigma = \{f, a, b\}$. We define this query by the automaton A_2 with $\mathsf{states}(A_2) = \{1, 2, *, y\}$ that will produce successful runs

[1] They use successful runs implicitly when defining the semantics of their pattern automata. Node selection is defined by pattern variables that are kept distinct from automata states

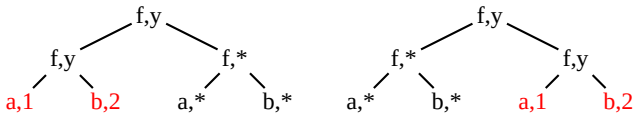

Fig. 2. Selecting pairs of a-leaves and next-sibling b-leaves: $\mathsf{q}^{\exists}_{A_2,\{(\mathbf{a},\mathbf{b})\}}$

of the form of Figure 2. The query is represented by $\mathsf{q}^{\exists}_{A_2,\{(1,2)\}}$. The automaton A_2 will assign state 1 to selected a-leaves and state 2 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. Every successful run of the automaton A_2 will select a single pair of nodes. The following rules verify these properties:

$$\begin{array}{llll} a\to 1 & b\to 2 & f(*,*)\to * & f(1,2)\to y \\ a\to * & b\to * & f(y,*)\to y & f(*,y)\to y \end{array}$$

This example illustrates the trick: different selected tuples are selected in different runs so that their components cannot be mixed up.

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

Sketch of proof. On the one hand, we can easily describe successful runs of tree automata in MSO. Existential run-based queries are thus definable in MSO. Let us prove now that every regular query is equal to some existential run-based query. Let $\mathsf{q}_{L(A)}$ 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 $1 \le i \le n$: $Q_i = \{(q, b_1, ..., b_n) \in \mathsf{states}(\mathsf{proj}(A)) \mid b_i = 1\}$. It remains to prove that $\mathsf{q}_{L(A)} = \mathsf{q}^{\exists}_{\mathsf{proj}(A),S}$. This follows from that for any term $t * \beta_1 * ... * \beta_n$ over $\Sigma \times \mathbb{B}^n$: $\mathsf{runs}_{\mathsf{proj}(A)}(t) = \{r * \beta_1 * ... * \beta_n \mid r \in \mathsf{runs}_A(t * \beta_1 * ... * \beta_n)\}$ and $\mathsf{succ\_runs}_{\mathsf{proj}(A)}(t) = \{r * \beta_1 * ... * \beta_n \mid r \in \mathsf{succ\_runs}_A(t * \beta_1 * ... * \beta_n)\}$.

Universal run-based n-ary queries quantify universally rather than existentially over successful runs. Universal n-ary queries were first introduced by Neven and Van den Bussche [18] in the framework of attribute grammars (universal BAGs and RAGs). In the monadic case, they are used by Frick, Grohe, and Koch [10].

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

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

This theorem has been proved for the monadic case [18] on basis of the two phase querying answering algorithm, which fails for the n-ary case. As we show here, the theorem generalizes to the n-ary case nevertheless.

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 $S \subseteq \mathsf{states}(A)^n$, $\mathsf{q}^{\forall}_{A,S} = (\mathsf{q}^{\exists}_{A,\mathsf{states}(A)^n \setminus S})^c$.

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

4 Query Answering

We consider the problems of enumerating all solutions or up to k solutions of run-based queries in a given tree t.

Proposition 1. *We can compute an existential run-based n-ary query $\mathsf{q}^{\exists}_{A,S}(t)$ in deterministic time $O(|S| * |A| * |t|^n)$ and hence in polynomial time for fixed n.*

Proof. The naive algorithm were to guess an n-tuple of nodes and test it for membership to $\mathsf{q}^{\exists}_{A,S}(t)$. By a deterministic algorithm this requires time $O(|S| * |A| * |t|^{n+1})$, so we need less naive algorithm. The idea of our algorithm is to guess a selection tuple $(p_1, \ldots, p_n) \in S$ and a tuple $(\pi_1, \ldots, \pi_{n-1}) \in \mathsf{nodes}(t)^{n-1}$ and to compute the last remaining node by answering a monadic query depending on the previous choices. Let $t^{p_1,\ldots,p_{n-1}}_{\pi_1,\ldots,\pi_{n-1}}$ be the tree over $\Sigma \cup (\Sigma \times \mathsf{states}(A))$ obtained from t by annotating the node labels of π_i by p_i for all $1 \leq i \leq n-1$.

Let $B(A)$ be the tree automaton with signature $\Sigma \cup (\Sigma \times \mathsf{states}(A))$ that operates like A except that maps all annotated nodes to their annotation. We define $\mathsf{states}(B(A))$ as $\mathsf{states}(A)$, $\mathsf{final}(B(A))$ as $\mathsf{final}(A)$ and $\mathsf{rules}(B(A))$ by:

$$\begin{aligned}\mathsf{rules}(B(A)) = \mathsf{rules}(A) \cup\ & \{(a,p) \to p \mid a \to p \in \mathsf{rules}(A)\} \\ & \cup \{(f,p)(p_1,p_2) \to p \mid f(p_1,p_2) \to p \in \mathsf{rules}(A)\}\end{aligned}$$

We can now compute $\mathsf{q}^{\exists}_{A,S}(t)$ on basis of the following representation:

$$\mathsf{q}^{\exists}_{A,S}(t) = \{(\pi_1, ..., \pi_{n-1}, \pi) \mid (p_1, \ldots, p_n) \in S,\ \pi \in \mathsf{q}^{\exists}_{B(A),\{p_n\}}(t^{p_1,\ldots,p_{n-1}}_{\pi_1,\ldots,\pi_{n-1}})\}$$

We have to answer $|S| * |t|^{n-1}$ monadic queries of the form $\mathsf{q}^{\exists}_{B(A),\{p_n\}}(t^{p_1,\ldots,p_{n-1}}_{\pi_1,\ldots,\pi_{n-1}})$ each of which requires linear time $O(|B(A)| * |t|)$. Note that the size of $|B(A)|$ is $2|A|$. Thus, the overall deterministic time complexity is $O(|S| * |A| * |t|^n)$.

The duality of existential and universal queries $\mathsf{q}^{\forall}_{A,S} = (\mathsf{q}^{\exists}_{A,\mathsf{states}(A)^n \setminus S})^c$ yields an analogous polynomial time complexity bound for answering universal n-ary queries $\mathsf{q}^{\forall}_{A,S}(t)$ with fixed tuple size n by $O((|\mathsf{states}(A)|^n - |S|) * |A| * |t|^n)$.

Proposition 2. *The emptiness problem of n-ary queries $\mathsf{q}^{\exists}_{A,S}(t) = \emptyset$ is NP-complete for unbounded n, i.e., if n belongs to the input of the problem, as well as the automaton A, the selection set $S \subseteq \mathsf{states}(A)$, and the tree t.*

Proof. The problem is clearly in NP: it suffices to guess a labeling of t by states of A and a selection tuple s from S; one can then check in $O(|A| * |t|)$ whether this labeling is a successful run and that each component of s labels at least one node in this run. Now, we give a polynomial reduction of CNF satisfiability into our problem. The idea is to associate with a given CNF formula ϕ a word w (which can be viewed as a unary tree) over the alphabet $\{\mathsf{x}, \mathsf{a}, \mathsf{n}, \mathsf{p}\}$ of the form $\mathsf{x}l_{11}l_{12}...l_{1n}...\mathsf{x}l_{k1}l_{k2}...l_{kn}$, where n is the number of clauses of ϕ and k the number of Boolean variables. A part $\mathsf{x}l_{i1}...l_{in}$ means that the i-th variable appears positively in the j-th clause if $l_{ij} = \mathsf{p}$, negatively if $l_{ij} = \mathsf{n}$ and does not appear if $l_{ij} = \mathsf{a}$. Then, we give the following rules for an automaton A with states $\{0,1\} \cup \{s^b_i, u^b_i \mid b \in \{0,1\}, 1 \leq i \leq n\}$:

$\mathsf{x} \to 0$	$\mathsf{x} \to 1$	$\mathsf{x}(\_) \to 0$	$\mathsf{x}(\_) \to 1$
$\mathsf{a}(b) \to u^b_1$	$\mathsf{a}(s^b_j) \to u^b_{j+1}$	$\mathsf{a}(u^b_j) \to u^b_{j+1}$	
$\mathsf{n}(0) \to s^0_1$	$\mathsf{n}(s^0_j) \to s^0_{j+1}$	$\mathsf{n}(u^0_j) \to s^0_{j+1}$	
$\mathsf{n}(1) \to u^1_1$	$\mathsf{n}(s^1_j) \to u^1_{j+1}$	$\mathsf{n}(u^1_j) \to u^1_{j+1}$	
$\mathsf{p}(0) \to u^0_1$	$\mathsf{p}(s^0_j) \to u^0_{j+1}$	$\mathsf{p}(u^0_j) \to u^0_{j+1}$	
$\mathsf{p}(1) \to s^1_1$	$\mathsf{p}(s^1_j) \to s^1_{j+1}$	$\mathsf{p}(u^1_j) \to s^1_{j+1}$	

where "_" denotes any state, $b \in \{0,1\}$ and $1 \leq j \leq n$. We accept all runs. Then the selection set is defined as $S_1 \times ... \times S_n$, with $S_i = \{s^b_i \mid 0 \leq b \leq 1\}$. As the size of the word is $(n+1) * k$ and the size of the automaton is in $O(n)$, the reduction is polynomial. There is a correspondence between runs of the automaton on w and truth assignments, and a run will be selecting iff the corresponding assignment satisfies all the clauses. The idea is to assign true (1) or false (0) value to a variable (represented by the x symbol) and to select all following clauses satisfied by the assignment. For example, if we consider $\psi = (x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge x_2$, then x p p a x n a p x a p a is its encoding, and $1\ s^1_1\ s^1_2\ u^1_3\ 1\ u^1_1\ u^1_2\ s^1_3\ 1\ u^1_1\ s^1_2\ u^1_3$ is a run selecting some n-tuples. So, ψ is satisfiable if and only if $\mathsf{q}^{\exists}_{A,S}(w) \neq \emptyset$.

5 Queries by unambiguous tree automata

We next study run-based n-ary queries by unambiguous tree automata. This is a subclass of tree automata with a restricted amount of nondeterminism.

A tree automaton A is *(bottom-up) deterministic* if no two of its rules have the same left hand sides. It is *unambiguous* if no tree permits more than one successful run by the automaton. Deterministic tree automata are clearly unambiguous, while unambiguous automata may be nondeterministic; they have multiple runs on the same tree of which at most one is successful.

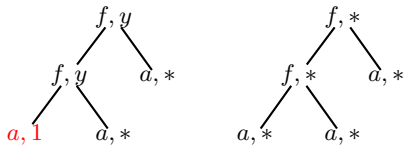

Fig. 3. Selecting left most leaves:$\mathsf{q}^{\exists}_{A_3,\{1\}}$. Only the left run of A_3 is successful.

Definition 4. *We call an n-ary query* unambiguous *(resp. deterministic) if it has the form $\mathsf{q}^{\exists}_{A,S}$ for some unambiguous (resp. deterministic) tree automaton A.*

Nondeterministic tree automata can recognize all regular language, but they an not define all MSO-definable queries in run-based fashion. A simple counter example is the monadic query that selects the left-most leaf in binary trees over $\Sigma = \{f, a\}$. It can be defined in run-based fashion as $\mathsf{q}^{\exists}_{A_3,\{1\}}$ by automaton A_3 which licences the runs in Fig. 3. Successful runs of A_3 label left most leaves by 1 and all others by $*$. They map ancestors of left most leaves to y and all other inner nodes to $*$. The final states are y and 1. This is done by the following states and rules:

$$\begin{array}{rlll} \mathsf{states}(A_3) = \{1, *, y\} & a \to 1 & f(1,*) \to y & f(y,*) \to y \\ \mathsf{final}(A_3) = \{1, y\} & a \to * & f(*,*) \to * & \end{array}$$

Automaton A_3 is not bottom-up deterministic, but unambiguous. Nondeterminism is needed in order to distinguish left most leaves from all others. When processing bottom-up, the automaton has to inspect the context, in order to decide whether a leaf is left-most. So it needs to guess this property for all leaves and then verify the correctness of the guesses later on. Correctness is proved by successful runs.

Proposition 3. *[[18, 3]] All monadic MSO-definable queries are unambiguous.*

Proof. We present a sketch of a proof based on Thatcher and Wright's theorem plus projection. Let $\phi(x)$ be MSO formula with one free variable x, which defines a monadic query in binary trees over Σ. We can express the same query by the following MSO formula with one free set variable p:

$$\mathsf{greatest}_{\phi}(p) = \forall x.p(x) \leftrightarrow \phi(x)$$

This formula requires to collect all possible values for x satisfying ϕ in p, so that p denotes the greatest of set containing nodes selected by $\phi(x)$. By Thatcher and Wright's Theorem 1 there exists a bottom-up deterministic tree automaton A that recognizes the tree language $L_{\mathsf{greatest}_{\phi}(p)}$, which contains all $\Sigma \times \mathbb{B}$ trees encoding models of $\mathsf{greatest}_{\phi}(p)$. The projection automaton $\mathsf{proj}(A)$ of A to Σ is unambiguous. To see this, note that the language of A is functional: for every Σ-tree t there exists at most one Boolean tree β such that $t \times \beta \in L(A)$. This holds since the value of β is determined by the result of the query by $\phi(x)$ on t. By determinism of A there is at most one successful run $r \in \mathsf{succ\_runs}_A(t \times \beta)$. Hence, there is at most one successful run $r \times \beta \in \mathsf{succ\_runs}_{\mathsf{proj}(A)}(t)$. Furthermore $\mathsf{q}_{L(A)} = \mathsf{q}_{\mathsf{proj}(A),\mathsf{states}(A)\times\{1\}}$.

Proposition 4. *Every deterministic monadic MSO defined query can be transformed effectively into a run-based query* $\mathsf{q}^{\exists}_{B,S}$ *by a deterministic automaton* B.

Proof. We proceed as in the proof of Proposition 3. Let A be a deterministic automaton recognizing $L_{\mathsf{greatest}_{\phi}(p)}$ and $\mathsf{proj}(A)$ is Σ-projection. We know that $\mathsf{proj}(A)$ is unambiguous and that it can express the query by $\phi(x)$. Furthermore, it can be checked that this automaton is deterministic after deleting unproductive states iff the query is deterministic.

5.1 Efficiency and expressiveness

We call an n-ary query *Cartesian closed* if it is a Cartesian product of monadic queries. If A is unambiguous then we can represent n-ary queries $\mathsf{q}^{\exists}_{A,S}$ as a finite unions of Cartesian closed queries:

$$\mathsf{q}^{\exists}_{A,S} = \bigcup_{(p_1,\ldots,p_n)\in S} \mathsf{q}^{\exists}_{A,\{p_1\}} \times \ldots \times \mathsf{q}^{\exists}_{A,\{p_n\}}$$

This holds, since all components of a tuple will be selected in the same successful run. We can use this representation of the answer set to enumerate answers of unambiguous queries on demand.

Proposition 5. *The emptyness probem* $\mathsf{q}^{\exists}_{A,S}(t) = \emptyset$ *can be solved in time* $O(n * |S| * |A| * |t|)$.

Proof. We compute the above representation of $\mathsf{q}^{\exists}_{A,S}(t)$. For all $(p_1, \ldots, p_n) \in S$ we compute $\mathsf{q}^{\exists}_{A,\{p_i\}}$ and check whether at least one of them is empty. We thus have to compute $O(n * |S|)$ answers to monadic queries each of them in time $O(|A| * |t|)$. Alltogether this requires time $O(n * |S| * |A| * |t|)$.

We can thus decide the emptyness of unambiguous n-ary queries in polynomial time even for unbounded n. This is in contrast to more general run-based n-ary queries by tree automata (Proposition 2).

Theorem 4. *Unambiguous* n*-ary queries capture the class of finite unions of Cartesian closed regular* n*-ary queries.*

Proof. We have already seen one direction. Next note that Cartesian closed regular queries are unambiguous. Indeed regular monadic queries are unambiguous by Proposition 3 and Cartesian products of unambiguous queries are clearly unambiguous too. It remains to prove that finite unions of unambiguous queries are unambiguous. Let $q = \cup_{j=1}^{k} \mathsf{q}^{\exists}_{A_i,S_i}$ 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, $q = \mathsf{q}^{\exists}_{A,S}$.

Finally, note that any unambiguous tree automata A_i can be made strictly unambiguous: let $\bar{A}_i$ be 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, $\mathsf{q}^{\exists}_{A'_i,S_i} = \mathsf{q}^{\exists}_{A_i,S_i}$.

Proposition 6. *A query is unambiguous iff it can be expressed by a Boolean combination (disjunction, conjunction and negation) of monadic MSO formulas.*

Proof. Using that regular and MSO-definable monadic queries coincide, by Theorem 4, an unambiguous n-ary query can be represented as a finite disjunction of formulas of the form $\phi_1(x_1) \wedge \ldots \wedge \phi_n(x_n)$, the ϕ_i's being monadic MSO formulas. Conversely, any Boolean combination of monadic MSO formulas can be turned into a finite disjunction of conjunction of monadic MSO formulas, and thus be represented as a finite union of Cartesian products of monadic regular queries.

5.2 Faithful MSO formulas

Unambiguity of a query will rely on existence of a faithful formula defining it, where faithful formulae are defined by:

Definition 5. *Let ϕ be a MSO formula with n free second-order variables $p_1, ..., p_n$.*

- *ϕ is $k-$*faithful *if* $\sup_{t \in T_\Sigma} |\{(\alpha(p_1), ..., \alpha(p_n)) \mid t, \alpha \models \phi\}| \leq k$.
- *ϕ is* faithful *if it is $k-$faithful for some k.*

Proposition 7. *ϕ is faithful iff it is equivalent to a finite disjunction of $1-$faithful formulae.*

Proof. More precisely, we prove that ϕ is $k-$faithful iff it is a finite disjunction of k $1-$faithful formulae. A finite disjunction of k $1-$faithful formulae is clearly $k-$faithful. Conversely let ϕ be a $k-$faithful formula. First, let us recall that the lexicographic ordering over $n-$uples is MSO definable by $lex(x_1, ...x_n, y_1, ..., y_n) =_{def} \vee_{k=1}^{n} (\wedge_{i=1}^{k-1} x_i = y_i \wedge x_k < y_k)$

Now, let us define a total ordering on $n-$uples of sets of nodes by

$le(p_1, ..., p_n, q_1, ...q_n) =_{def} \wedge_{i=1}^{n} p_i = q_i \vee \exists x_1, ..., x_n \wedge_{i=1}^{n} p_i(x_i) \wedge \vee_{i=1}^{n} \neg q_i(x_i) \wedge \forall y_1, ..., y_n \; lex(y_1, ...y_n, x_1, ..., x_n) \rightarrow \wedge_{i=1}^{n} [p_i(y_i) \leftrightarrow \wedge_{i=1}^{n} q_i(y_i)]$.

Last, we define a family of $1-$faithful formulae ϕ_i, $1 \leq i \leq k$ by:

$\phi_i(p_1, ..., p_n) =_{def} \phi(p_1, ..., p_n) \wedge \wedge_{j=1}^{i-1} \neg \phi_j(p_1, ..., p_n) \wedge$
$\forall q_1, ..., q_n (\phi(q_1, ..., q_n) \wedge (\wedge_{j=1}^{i-1} \neg \phi_j(q_1, ..., q_n)) \rightarrow le(p_1, ..., p_n, q_1, ...q_n))$

It is easy to check that the ϕ_i are $1-$faithful and, as ϕ is $k-$faithful, ϕ is equivalent to $\vee_{i=1}^{n} \phi_i$.

Proposition 8. *A regular n-ary query is*

1. *Cartesian closed iff it can be defined by some $1-$faithful formula.*
2. *unambiguous iff it can be defined by some faithful formula.*

Proof. Let q a regular Cartesian closed query defined by ϕ. Let us define $\phi_i(x)$ by $\exists x_1, ..., x_{i-1}, x_{i+1}, ..., x_n, \phi(x_1, ..., x_{i-1}, x, x_{i+1}, ..., x_n)$. Then q can be defined by the $1-$faithful formula $\forall x \wedge_{i=1}^{n} (p_i(x) \leftrightarrow \phi_i(x))$

Conversely, if q is defined by a $1-$faithful formula, q is clearly Cartesian closed.

The rest of the proposition is then directly obtained by Proposition 7 and Theorem 4. Furthermore, as proofs of Proposition 7 and Theorem 4 are effective, given a query q defined by a formula ϕ and knowing that ϕ is faithful, we can effectively construct (A, S) computing the query q, with A unambiguous.

5.3 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 this property is close to independence of variables in constraint databases [14, 8]; however here we consider an infinite collection of finite tree structures, instead of one fixed structure.

Note that deciding whether a regular query is Cartesian closed is straightforward as it can be defined in MSO. Similarly by using construction of Proposition 7, we can decide $k-$faithfulness of a MSO formula, for a given k. However, deciding whether a regular query is a finite union of Cartesian closed regular queries requires more sophisticated techniques. First, given a query q, we construct a formula which is faithful iff q is unambiguous. Second, we prove how to decide faithfulness of a formula.

Let q a query defined by the (MSO) formula $\phi_q(x_1, \ldots, x_n)$. We will define ϕ_q^{max}, a MSO formula defining q with good compactness properties: it will be faithful as soon as q can be defined by a faithful formula. Roughly speaking, given a tree t, t, α will model ϕ_q^{max} iff it is correct ($\alpha(p_1) * ... * \alpha(p_n)$ is included in $q(t)$) and maximal (no node can be added to one $\alpha(p_i)$ while keeping correct). $\phi_q^{\mathsf{max}}(p_1, \ldots, p_n)$ will be the following formula:

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

Lemma 3. *A query q is a finite union of Cartesian closed queries iff ϕ_q^{max} is faithful.*

Proof. By Proposition 8 we just have to prove that if the query q is a finite union of Cartesian closed queries, then ϕ_q^{max} is faithful. Let q be a finite union of Cartesian closed queries. There exists some natural number k s.t. $q = \cup_{j=1}^{k} q_j^1 \times \ldots \times q_j^n$, each q_j^i being a monadic 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 w.r.t. q. Then, let π and π' be two nodes. If for each $1 \leq j \leq k$, π belongs to $q_j^i(t)$ iff π' belongs to $q_j^i(t)$, then $\pi \equiv_i \pi'$. This implies that $\equiv_i$ is of finite index bounded by 2^k.

Now let t and α such that $t, \alpha \models \phi_q^{\mathsf{max}}$. Let π be one node selected in the i-th position, i.e. belonging to $\alpha(p_i)$. Then, by maximality of ϕ_q^{max}, if $\pi \equiv_i \pi'$ then π'

belongs also to $\alpha(p_i)$. This implies that $\alpha(p_i)$ is a union of equivalence classes for $\equiv_i$. So, the cardinality of the set $\{(\alpha(p_1), ..., \alpha(p_n)) \mid t, \alpha \models \phi_q^{\mathsf{max}}\}$ is upper-bounded by $2^{n.2^k}$.

Let us note that if ϕ_q^{max} is faithful as soon there is a faithful formula defining q, it is non necessarly the "most faithful" one or the "less redundant" one. Indeed let us suppose that q is defined by $\vee_{i=1}^{2} r_i(x_1) \wedge s_i(x_2)$ for some r_i, s_i. q is clearly $2-$faithful whereas in ϕ_q^{max}, valuation associated with $(\wedge r_i, \vee s_i)$ or $(\vee r_i, \wedge s_i)$ would be added.

Now, let q be a regular query (given by a tree automator or a formula): first we construct ϕ_q^{max} and A a deterministic automaton recognizing the tree language over $\Sigma \times \mathbb{B}^n$ $L_{\phi_q^{\mathsf{max}}(p_1,...,p_n)}$. Then, we compute an automaton $\mathsf{proj}(A)$ as in Theorem 2. Clearly the number of accepting runs on t in $\mathsf{proj}(A)$ is the cardinal of $\{(\alpha(p_1), ..., \alpha(p_n)) \mid t, \alpha \models \phi_q^{\mathsf{max}}\}$.

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 [20]). *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 provides a procedure for deciding ambiguity of q. Furthermore, this gives a way to compute an unambiguous automaton computing q. Indeed, by proposition 8, as soon as we know that ϕ_q^{max} is faithful, we can compute, from an automaton or a formula defining q, (B, S) with B an unambiguous automaton s.t. $q = \mathsf{q}_{B,S}^{\exists}$.

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

Note that the construction of (B, S) could also be done by eliminating directly ambiguity from $\mathsf{proj}(A)$ defined above. Indeed, let B be an automaton whose ambiguity degree is at most k. We can build an automaton B_k simulating B on trees which have at least k accepting runs in B (by making the product of k copies of B and checking the k runs are different); as the degree of B is k, B_k will be unambiguous. Then, you can build an unambiguous automaton B_{k-1} simulating B on trees which have exactly $k-1$ accepting runs in B, by a similar construction and checking that the tree is not accepted by B_k. By iterating the construction, you can build $(B_i, S_i)_{i=1}^{k}$, with B_i unambiguous automata simulating B on trees which have exactly i accepting runs in B: q is the union of the corresponding queries and by using effective closure under union, you can then build an unambiguous automaton for q.

6 Querying unranked trees

Our results carry over to automata for unranked trees, in particular to the unranked tree automata (UTAs) of Brüggemann, Klein, and Wood [4], where horizontal tree languages are represented by finite word automata.

An unranked tree is built 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 *UTA* 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 UTAs 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 UTAs (in contrast to bottom-up determinism [16]). The same holds for the notions of run-based queries by UTAs.

Theorem 7. *Existential and universal n-ary queries by runs of unranked tree automata capture MSO over unranked trees (comprising the* $\mathsf{next\_sibl}$*-relation). Run-based queries by unambiguous UTAs capture the class of finite unions of Cartesian closed queries. This property is decidable.*

We only give a sketch of the proof. The main idea is to convert queries by UTAs 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 UTAs 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.

Acknowledgements. Thanks to the anonymous referees for the reference to L. Libkin's work [14] and acknowledge discussions with F. Neven, W. Martens, and T. Schwentick.

References

1. R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *28th International Conference on Very Large Data Bases*, 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. *Journal of Comput. and Syst. Sci.*, 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*, volume 3264 of *Lecture Notes in Artificial Intelligence*, pages 91–102. Springer Verlag, 2004.

6. J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In *19th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 105 – 118. Springer Verlag, 2004.
7. G. Castagna. Patterns and types for querying XML. In *10th International Symposium on Database Programming Languages*, Lecture Notes in Computer Science. Springer Verlag, Aug. 2005.
8. J. Chomicki, D. Q. Goldin, and G. M. Kuper. Variable independence and aggregation closure. In *ACM Conference on Principle of Databases*, pages 40–48, 1996.
9. 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.
10. M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees. In *18th IEEE Symposium on Logic in Computer Science*, pages 188–197, 2003.
11. G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proceedings of the* 17^{th} *LICS*, Lecture Notes in Computer Science, pages 189–202, Copenhagen, 2002.
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 *ACM Symposium on Principles of Database Systems*. ACM-Press, 2004.
13. H. Hosoya and B. Pierce. Regular expression pattern matching for xml. *Journal of Functional Programming*, 6(13):961–1004, 2003.
14. L. Libkin. Variable independence for first-order definable constraints. *ACM Transactions on Computational Logics*, 4(4):431–451, 2003.
15. S. Maneth, A. Berlea, T. Perst, and H. Seidl. Xml type checking with macro tree transducers. In *24th ACM Symposium on Principles of Database Systems*, pages 283–294, New York, NY, USA, 2005. ACM-Press.
16. W. Martens and J. Niehren. Minimizing tree automata for unranked trees. In *10th International Symposium on Database Programming Languages*, Lecture Notes in Computer Science. Springer Verlag, Aug. 2005.
17. M. Marx. Conditional XPath, the first order complete XPath dialect. In *Proceedings of the symposium on Principles of database systems*, pages 13–22, 2004.
18. 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.
19. F. Neven and T. Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.
20. H. Seidl. On the finite degree of ambiguity of finite tree automata. *Acta Informatica*, 26(6):527–542, 1989.
21. 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.