

Learning n -ary Node Selecting Tree Transducers from Completely Annotated Examples

A. Lemay¹, J. Niehren², and R. Gilleron¹
Mostrare project of INRIA Futurs, LIFL, Lille France
¹ University of Lille 3 ² INRIA Futurs

Abstract. We present the first algorithm for learning n -ary node selection queries in trees from completely annotated examples by methods of grammatical inference. We propose to represent n -ary queries by deterministic n -ary node selecting tree transducers (n-NSTTs). These are tree automata that capture the class of monadic second-order definable n -ary queries. We show that n-NSTT defined polynomially bounded n -ary queries can be learned from polynomial time and data. An application in Web information extraction yields encouraging results.

1 Introduction

The problem of selecting nodes in trees is the most basic and fundamental querying problem in the context of XML [8,14,12]. In this paper, we propose a new machine learning algorithm based on grammatical inference for learning n -ary node selection queries. We will illustrate its interest in an application to wrapper induction for Web information extraction [10,4,13,18].

We consider finite rooted directed sibling-ordered unranked trees $t \in T_\Sigma$ with nodes labeled in a fixed signature Σ . An n -ary query in such trees [14,9,15] is a function q that maps trees $t \in T_\Sigma$ to sets of n -tuples of nodes $q(t) \subseteq \text{nodes}(t)^n$. Boolean queries are 0-ary queries and can be identified with tree languages¹. Monadic queries where $n = 1$ select nodes in trees. Binary queries where $n = 2$ select pairs of nodes in trees, and so on. The most natural way to represent n -ary queries is *monadic second-order logic (MSO)*, i.e. by MSO-formulas with n free variables. MSO-defined queries are *regular*, i.e. definable by tree automata over $\Sigma \times \text{Bool}^n$, and vice versa. This follows from Thatcher and Wright's theorem in the case of ranked trees [19] and carries over to unranked trees.

We investigate learning algorithms for MSO-definable n -ary queries. The input is a set of completely annotated examples for the target query q . These are pairs $(t, q(t))$ for some tree $t \in T_\Sigma$. Completely annotated examples contain positive information on all tuples in $q(t)$, and negative information on all others. In the Boolean case, they coincide with the positive and negative examples for tree languages, i.e. whether a tree belongs to the language or not.

¹ This is well-known in database theory. A tree t belongs to the language defined by a Boolean query q if and only if the empty 0-tuple $()$ belongs to $q(t)$.

All learnability results depend on how n -ary queries are represented. The following properties are wishful in general, and in particular for applications to Web information extraction.

Learnability For all n -ary queries q a representative can be learned from polynomial time and data in form of completely annotated examples.

Expressiveness All n -ary MSO-definable queries can be represented.

Efficiency Given a representation of an n -ary query q and a tree t the set $q(t)$ can be enumerated efficiently.

For $n = 0$ all three conditions can be satisfied when representing tree languages by bottom-up deterministic tree automata. Completely annotated examples then coincide with positive and negative examples. Learning algorithms for deterministic tree automata from positive and negative examples (RPNI) have been studied in [5].

For $n = 1$, these properties have been shown recently [1,2] when representing monadic queries by deterministic *node selecting tree transducer* (NSTTs). These are *functional tree automata* over $\Sigma \times \mathbf{Bool}$, which define relabeling functions from trees over Σ to trees over \mathbf{Bool} . Selected nodes are relabeled to true, all others to false. A learning algorithm from polynomial time and data can be obtained by adapting RPNI to deterministic NSTTs while taking functionality into account, for the treatment of negative information. MSO completeness for deterministic NSTTs can still be inferred from Thatcher and Wright's theorem, despite of the restriction to functionality. Efficient query answering is possible in linear time by a two phases algorithm.

For $n > 1$, the question is still open whether there exists a representation formalism for n -ary queries that satisfies the above three properties. A number of principle problems arise. The most disturbing fact is that functional tree automata over $\Sigma \times \mathbf{Bool}^n$ are not sufficiently expressive for $n > 1$. They can only define finite unions of Cartesian closed n -ary queries as shown in [15]. These are clearly insufficient in theory and practice.

Furthermore, the number of n -tuples in $q(t) \subseteq \mathbf{nodes}(t)^n$ may become exponential for unbounded n so that efficient enumeration becomes an issue for $n > 1$. Completely annotated examples for q may thus become huge. This should not happen in practice of information extraction. In theory, we will restrict ourselves to queries where the number of answers is polynomially bounded in the size of the tree. Our learning algorithms will have to use compact representations for huge sets of negative examples, i.e., complements $\mathbf{nodes}(t)^n - q(t)$.

In this article, we propose to represent n -ary queries in Σ -trees by deterministic tree automata over $\Sigma \times \mathbf{Bool}^n$ that recognize canonical languages, where every accepted tree corresponds to precisely one n -tuple. We call tree automata with canonical languages *n -ary node selection tree transducer* (n -NSTTs).

All tree automata obtained from MSO formula have canonical languages as long as all free variables are first-order. However, most NSTTs are *not* 1-NSTTs and vice versa. Despite of this, both classes of automata have the same expressiveness – they can both represent all monadic MSO definable queries, but

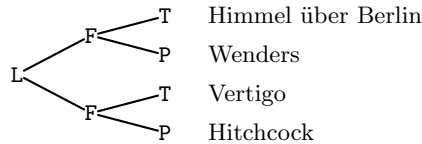


Fig. 1. The binary tree `two-films` with some data content.

differently. We show how to learn deterministic n-NSTTs from completely annotated examples. Our algorithm satisfies the learning model from polynomial time and data, under the assumption that the number of answers to queries is polynomially bounded in the size of the tree. The main problem is to represent the possibly exponential amount of negative information contained in a set of completely annotated examples in a compact manner. In the monadic case, this could be solved by the functionality requirement on NSTTs which is no more available for n-NSTTs. We also show that answers of n-ary queries represented by deterministic n-NSTTs can be enumerated efficiently.

We have implemented our algorithm and started applying it to Web information extraction. We assume highly structured Web pages generated by some database. First experiments yield encouraging results for the n-ary case, that let us hope for competitive systems by future work.

2 N-ary Node Selecting Tree Transducer

We introduce n-NSTTs for binary trees. Unranked trees will be considered in Section 6. The principal difference between 1-NSTTs as presented here and NSTTs from [1] is essential to generalize smoothly from monadic to n-ary queries.

Let $\mathbb{N} = 1, 2, \dots$ be the natural numbers without 0 and $\mathbf{Bool} = \{0, 1\}$ the Booleans. We denote the cardinality of a set A by $|A|$. Given a finite set Σ of *node labels*, a finite directed sibling-ordered *binary tree* $t \in T_\Sigma$ is either a label $a \in \Sigma$ or a triple $a(t_1, t_2)$ consisting of a label $a \in \Sigma$ and two binary trees $t_1, t_2 \in T_\Sigma$. Fig. 1, for instance, contains the binary tree `two-films` = $L(F(T, P), F(T, P))$ where $\Sigma = \{L, F, T, P\}$. This tree represents a list (L) of two films (F) each having a title (T) and a producer (P). Rather than putting data content into tree labels, we assume an external mapping from nodes to data values. Note that nodes may carry the same label while containing different data. For instance, both films have different producers and titles. This works as long as we carefully distinguish different nodes with the same label.

We identify nodes of trees with their relative address from the root. The node 2·1, for instance, is the first child of the second child of the root. In the example in Fig. 1, this is the T node containing *Vertigo*. We write $t(v)$ for the label of some $v \in \mathbf{nodes}(t)$, for instance: `two-films`(2·1) = T. We denote by $\mathbf{nodes}(t) \subseteq \mathbb{N}^*$ the set of nodes of a tree t . We say that two trees have *the same shape* if they have the same sets of nodes. We write $\mathbf{size}(t)$ for $|\mathbf{nodes}(t)|$.

Definition 1. An n -ary query in trees over Σ is a function q from trees $t \in T_\Sigma$ to sets of n -tuples of nodes $q(t) \subseteq \mathbf{nodes}(t)^n$.

Let the binary query `title-producer-pairs` ask for all pairs of titles and producers in trees that encode lists of films. From the tree `two-films`, this query selects the following node pairs: `title-producer-pairs(two-films) = {(1·1, 1·2), (2·1, 2·2)}`

The usual idea how to represent n -ary queries by tree automata stems from early work on MSO [19]. It consists in identifying n -ary queries over Σ with tree languages over $\Sigma \times \mathbf{Bool}^n$. These can then be recognized by a tree automaton. There are several possibilities in doing so, which differ in how many n -tuples may be encoded by Boolean annotations at the same tree. For $n = 2$ for instance, consider $L_{00}(F_{00}(T_{10}, P_{01}), F_{00}(T_{10}, P_{01}))$. This tree is annotated by pairs of Booleans that represent 4 pairs of nodes: $\{(1·1, 1·2), (2·1, 2·2), (1·1, 2·2), (2·1, 1·2)\}$. The third and fourth pair may be unwanted since they mix up the titles and producers. We cannot annotate, however, only the first two pairs to the same copy of tree `two-films`, we need two independent copies: $L_{00}(F_{00}(T_{10}, P_{01}), F_{00}(T_{00}, P_{00}))$ and $L_{00}(F_{00}(T_{00}, P_{00}), F_{00}(T_{10}, P_{01}))$. This contrasts strongly with the monadic case, where one can always annotate all tuples in $q(t)$ to a unique copy of t . Such compact annotations lead to functional tree languages, as recognized by the NSTTs in [1].

In the n -ary case, however, several copies of t need to be annotated, one for each of its n -tuples. We call trees over $\Sigma \times \mathbf{Bool}^n$ *tuple trees* if they are annotated by a single n -tuple. Every tree over $\Sigma \times \mathbf{Bool}^n$ can be decomposed in a unique manner into two trees of the same shape, a tree $t \in T_\Sigma$ and its Boolean annotation $\beta \in T_{\mathbf{Bool}^n}$. We write $t \times \beta$ for the unique tree in $\Sigma \times \mathbf{Bool}^n$ that can be decomposed into t and β . Given a n -tuple α and $1 \leq i \leq n$ let $\Pi_i(\alpha)$ be the i -th component of α . If $t \times \beta$ is a tuple tree then β corresponds to a unique n -tuple $\beta \in \mathbf{nodes}(t)^n$ such that:

$$\forall v \in \mathbf{nodes}(t) \forall 1 \leq i \leq n : \Pi_i(\beta(v)) = 1 \text{ iff } \Pi_i(\beta) = v$$

We call tree languages over $\Sigma \times \mathbf{Bool}^n$ *canonical* if all trees contained are tuple trees. Clearly, every n -ary query q over Σ is represented by exactly one canonical language.

We will use tree automata to represent canonical languages, so we recall their definition. A *tree automaton* A over Σ is a triple that consists of three finite sets $\mathbf{states}(A)$, $\mathbf{final}(A) \subseteq \mathbf{states}(A)$, and $\mathbf{rules}(A)$, so that all rules are or the form $a \rightarrow q$ or $a(q_1, q_2) \rightarrow q$ where $a \in \Sigma$ and $q, q_1, q_2 \in \mathbf{states}(A)$. A *run* of a tree automaton A on a tree t is a function $r : \mathbf{nodes}(A) \rightarrow \mathbf{states}(A)$ so that all states that r assigns to nodes of t are justified by some rule of A . A run r of A on t is *successful* if it maps the root of t to a final state of A , i.e. $r(\varepsilon) \in \mathbf{final}(A)$. We write $\mathbf{succ_runs}_A(t)$ for the set of successful runs by A on t . The *language* $L(A)$ is the set of all trees $t \in T_\Sigma$ that permit a successful run by A . The size measure for automata in this paper counts states and rules: $\mathbf{size}(A) = |\mathbf{rules}(A)| + |\mathbf{states}(A)|$. We call a tree automaton *trimmed* if all of its states are used in some successful run.

Definition 2. An n -ary node selecting tree transducer (n-NSTT) over Σ is a tree automaton over $\Sigma \times \text{Bool}^n$ that recognizes a canonical language.

An n-NSTT A over Σ represents the n -ary query q_A in trees $t \in T_\Sigma$ such that:

$$q_A(t) = \{\beta \in \text{nodes}(t)^n \mid t \times \beta \in L(A)\}$$

In other words, a query q is represented by all n-NSTTs that recognize the language of all tuple trees for q . All such n-NSTTs are *equivalent* in that they recognize the same language. Thatcher and Wright's theorem [19] states that n-NSTTs capture the class of MSO-definable n -ary queries. Thus, 1-NSTT \neq NSTT even though both of them capture monadic MSO-definable queries.

3 Membership Testing to the Class of n-NSTTs

We present an efficient algorithm for testing whether a tree automaton is an n-NSTT. The results on types obtained on the way will help avoiding such tests during query induction in Section 5.

An n -type \mathbf{b} is an n -tuple of non-negative integers, that is $\mathbf{b} \in (\mathbb{N} \cup \{0\})^n$. All bit vectors in Bool^n are n -types. The *type of a tree* $\beta \in T_{\text{Bool}^n}$ is the n -type obtained by summing up all labels of nodes in β .

$$\mathsf{T}(\beta) = \sum_{v \in \text{nodes}(\beta)} \beta(v)$$

Note that $t \times \beta$ is a tuple tree if and only if $\mathsf{T}(\beta) = (1, \dots, 1) = 1^n$. Let A be a tree automaton over $\Sigma \times \text{Bool}^n$. To every $q \in \text{states}(A)$, we assign a set $\mathsf{T}(q)$ of n -types by the following inference rules:

$$\frac{(a, \mathbf{b}) \rightarrow q \in \text{rules}(A)}{\mathbf{b} \in \mathsf{T}(q)} \quad \frac{(a, \mathbf{b})(q_1, q_2) \rightarrow q \in \text{rules}(A) \quad \mathbf{b}_1 \in \mathsf{T}(q_1) \quad \mathbf{b}_2 \in \mathsf{T}(q_2)}{\mathbf{b} + \mathbf{b}_1 + \mathbf{b}_2 \in \mathsf{T}(q)}$$

Lemma 1. If r is a run of A on $t \times \beta$ then $\mathsf{T}(\beta) \in \mathsf{T}(r(\varepsilon))$.

Lemma 2. For all $q \in \text{states}(A)$ and $\mathbf{b} \in \mathsf{T}(q)$ there exists a tree $t \times \beta$ over $\Sigma \times \text{Bool}^n$ and a run r of A on this tree such that $q = r(\varepsilon)$ and $\mathsf{T}(\beta) = \mathbf{b}$.

Lemma 3. If A is a trimmed n-NSTT then $\mathsf{T}(q) \subseteq \text{Bool}^n$ is a singleton.

Proof. To see that $\mathsf{T}(q) \neq \emptyset$, note that we assume A to be trimmed. Thus there exists a tree $t \times \beta$ and a run r on that tree such that $r(\varepsilon) = q$. By Lemma 1 it follows that $\mathsf{T}(\beta) \in \mathsf{T}(q)$. To see that $\mathsf{T}(q) \in \text{Bool}^n$, let $\mathbf{b} \in \mathsf{T}(q)$. By Lemma 2 there exists a tree $t \times \beta$ over $\Sigma \times \text{Bool}^n$ and a run r of A on this tree such that $q = r(\varepsilon)$ and $\mathsf{T}(\beta) = \mathbf{b}$. Since A is trimmed there exists a tree in $\tilde{t} \times \tilde{\beta} \in L(A)$ that contains $t \times \beta$ as a subtree. Hence: $\mathbf{b} = \mathsf{T}(\beta) \leq \mathsf{T}(\tilde{\beta}) = 1^n$. It remains to show that $\mathsf{T}(q)$ is a singleton, so let us assume that $\mathbf{b}' \in \mathsf{T}(q)$ too. By Lemma 2 there exists a second tree $t' \times \beta'$ over $\Sigma \times \text{Bool}^n$ and a run r' of A on this tree such that $q = r'(\varepsilon)$ and $\mathsf{T}(\beta') = \mathbf{b}'$. Let $\tilde{t}' \times \tilde{\beta}'$ be the tree obtained by replacing one occurrence of $t \times \beta$ in $\tilde{t} \times \tilde{\beta}$ by $t' \times \beta'$. Note that $\tilde{t}' \times \tilde{\beta}' \in L(A)$, hence

$\tau(\beta') = 1^n$. Let V be the set of nodes of $\tilde{t} \times \tilde{\beta}$ that have not been affected by the substitution.

$$\begin{aligned} 1^n &= \tau(\tilde{\beta}) = \tau(\beta) + \sum_{v \in V} \tilde{\beta}(v) \\ 1^n &= \tau(\tilde{\beta}') = \tau(\beta') + \sum_{v \in V} \tilde{\beta}'(v) \end{aligned}$$

Since $\tilde{\beta}(v) = \tilde{\beta}'(v)$ for all $v \in V$, $\tau(\beta) = \tau(\beta')$ so that $\mathbf{b} = \mathbf{b}'$.

Lemma 4. *A trimmed automaton A over $\Sigma \times \text{Bool}^n$ is an n -NSTT iff $\tau(q) = \{1^n\}$ for all $q \in \text{final}(A)$.*

Proof. let A be a trimmed n -NSTT and let $q \in \text{final}(A)$. Since A is trimmed there exists a tree $t \times \beta$ and a run r on that tree such that $r(\varepsilon) = q$. Thus $t \times \beta \in L(A)$ so that $\tau(\beta) = 1^n$. By Lemma 1, it follows that $1^n \in \tau(q)$. This set is a singleton by Lemma 3 so that $\tau(q) = \{1^n\}$. For the converse, it follows from Lemma 1, that all $t \times \beta \in L(A)$ satisfy $\tau(\beta) = 1^n$ so that they are tuple trees.

Proposition 1. *Whether a tree automaton A over $\Sigma \times \text{Bool}^n$ is an n -NSTT can be decided in polynomial time $O(\text{size}(A) \times n)$. If so, all types in $\{\tau(q) \mid q \in \text{states}(A)\}$ can be computed in the same time.*

Proof. Lemma 4 gives us a way to check whether a tree automaton is an n -NSTT. In the first step, we trim the automaton without changing its language. This requires linear time $O(\text{size}(A) \times n)$. We then compute all values $\tau(q)$ by saturation with respect to the defining rules. We exit saturation immediately, if it tries to add a second element to some type set, or if it tries to add a non-Boolean n -type. If this happens then we return false, which is justified by Lemma 3. Note that all positions in rules will be touched at most once and all type sets at most twice. Hence, saturation can be implemented in time $O(\text{size}(A) \times n)$. If saturation succeeds then we apply the third step. All types have been computed successfully now. We check for all $q \in \text{final}(A)$ whether $\tau(q) = \{1^n\}$. If so we return true otherwise false, which is licenced by Lemma 4. This can be done in time $O(\text{size}(A) \times n)$ too.

4 Efficient Answer Enumeration

We develop an efficient algorithm for enumerating the answers of an n -NSTT defined query on a given input tree. The insights gained will again be used in our learning algorithm.

Given an n -NSTT A and a tree t the problem is to compute all β such that $t \times \beta \in L(A)$. The first step to do is to project A to a tree automaton over Σ that we denote by $\pi(A)$. This automaton satisfies $\text{states}(\pi(A)) = \text{states}(A)$ and $\text{final}(\pi(A)) = \text{final}(A)$. Its rules are inferred by the following two schemata where $a \in \Sigma$ and $\mathbf{b} \in \text{Bool}^n$:

$$\frac{(a, \mathbf{b})(q_1, q_2) \rightarrow q \in \text{rules}(A)}{a(q_1, q_2) \rightarrow q \in \text{rules}(\pi(A))} \qquad \frac{(a, \mathbf{b}) \rightarrow q \in \text{rules}(A)}{a \rightarrow q \in \text{rules}(\pi(A))}$$

Given an trimmed n -NSTT A , let $T_A : \mathbf{states}(A) \rightarrow \mathbf{Bool}^n$ be the function that maps states of A to their unique n -type according to Lemma 3. The following lemma permits to type rules of projections of n -NSTTs.

Lemma 5. *For all trimmed n -NSTTs A , labels $a \in \Sigma$, and $q, q_1, q_2 \in \mathbf{states}(A)$:*

$$\begin{aligned} a \rightarrow q \in \mathbf{rules}(\pi(A)) & \text{ iff } (a, T_A(q)) \rightarrow q \in \mathbf{rules}(A) \\ a(q_1, q_2) \rightarrow q \in \mathbf{rules}(\pi(A)) & \text{ iff } (a, \mathbf{b})(q_1, q_2) \rightarrow q \in \mathbf{rules}(A) \\ & \text{ where } \mathbf{b} = T_A(q) - T_A(q_1) - T_A(q_2) \end{aligned}$$

Proof. The implications from the right to the left are obvious from the definition of the rules of $\pi(A)$. For the converse there are two cases. First, assume $a \rightarrow q \in \mathbf{rules}(\pi(A))$. By definition of $\pi(A)$ there exists $\mathbf{b} \in \mathbf{Bool}^n$ such that $(a, \mathbf{b}) \rightarrow q \in \mathbf{rules}(\pi(A))$. Lemma 1 shows that $\mathbf{b} = T_A(q)$. Second, assume $a(q_1, q_2) \rightarrow q \in \mathbf{rules}(\pi(A))$. By definition of $\pi(A)$ there exists $\mathbf{b} \in \mathbf{Bool}^n$ such that $(a, \mathbf{b})(q_1, q_2) \rightarrow q \in \mathbf{rules}(A)$. Since A is trimmed, there exist a tree $t_1 \times \beta_1$ and $t_2 \times \beta_2$ over $\Sigma \times \mathbf{Bool}$ that can be evaluated by A into states q_1 and q_2 respectively. Thus, the tree $(a, \mathbf{b})(t_1 \times \beta_1, t_2 \times \beta_2)$ can be evaluated to q by A . Lemma 1 shows that $\mathbf{b} + T_A(q_1) + T_A(q_2) = T_A(q)$. Hence, $\mathbf{b} = T_A(q) - T_A(q_1) - T_A(q_2)$.

For every tree run r of a trimmed tree automaton A over $\Sigma \times \mathbf{Bool}^n$ on some tree with node v we define a function mapping nodes to n -types.

$$T_A^r(v) = \begin{cases} T_A(r(v)) & \text{if } v \text{ is a leaf} \\ T_A(r(v)) - T_A^r(r(v.1)) - T_A^r(r(v.2)) & \text{else} \end{cases}$$

Note that T_A^r can be identified with the unique $\beta \in T_{\mathbf{Bool}^n}$ such that $\beta(v) = T_A^r(v)$ for all nodes v .

Lemma 6. *For all trimmed n -NSTTs A , $t \in T_\Sigma$, and $r : \mathbf{nodes}(t) \rightarrow \mathbf{states}(A)$:*

$$r \in \mathbf{succ_runs}_{\pi(A)}(t) \text{ iff } r \in \mathbf{succ_runs}_A(t \times T_A^r)$$

Proof. Straightforward from Lemma 5 by induction on trees t .

Recall that a tree automaton is unambiguous, if no tree permits more than one successful run. All deterministic tree automata are unambiguous.

Proposition 2. *Let A be a trimmed unambiguous n -NSTT. For all trees $t \in T_\Sigma$, the function mapping $r \in \mathbf{succ_runs}_{\pi(A)}(t)$ to Boolean annotations T_A^r in $T_{\mathbf{Bool}^n}$ is a bijection with range $\{\beta \mid t \times \beta \in L(A)\}$.*

Proof. First note that the function always maps to $\{\beta \mid t \times \beta \in L(A)\}$. This follows from Lemma 6. If $r \in \mathbf{succ_runs}_{\pi(A)}(t)$ then $r \in \mathbf{succ_runs}_A(t \times T_A^r)$ so that $t \times T_A^r \in L(A)$. Second, we show that the function is onto. To see this, we show by induction on t that if r is a run of A on $t \times \beta$ then $\beta = T_A^r$. Let β such that $t \times \beta \in L(A)$. Thus, there exists $r \in \mathbf{succ_runs}_A(t \times \beta)$ so that $\beta = T_A^r$. By Lemma 6, it also holds that $r \in \mathbf{succ_runs}_{\pi(A)}(t)$ so that T_A^r is a value taken by the function. Third, we have to show that the function is one-to-one. Let $r_1, r_2 \in \mathbf{succ_runs}_{\pi(A)}(t)$ such that $T_A^{r_1} = T_A^{r_2}$. By Lemma 6 it holds that $r_i \in \mathbf{succ_runs}_A(t \times T_A^{r_i})$ for both $i = 1, 2$. Hence, r_1, r_2 are successful runs of A on the same tree, so they are equal by unambiguity of A .

Theorem 1. *For every unambiguous n-NSTT A , we can compute an algorithm in time $O(\mathbf{size}(A) \times n)$ that enumerates $q_A(t)$ with delay $O(\mathbf{size}(A) \times \mathbf{size}(t) \times n)$ per n -tuple.*

Hence, one can compute the answer set $q_A(t)$ for unambiguous n-NSTTs A on trees t in time $O((|q_A(t)| + 1) \times \mathbf{size}(A) \times \mathbf{size}(t) \times n)$.

Proof. In order to enumerate the answer set $q_A(t)$ of an n-NSTTs A it is sufficient to enumerate the set $\{\beta \mid t \times \beta \in L(A)\}$ since every β can be transformed in linear time into a unique n -tuple β by definition of n-NSTTs. This set is in bijection to the set of successful runs of $\pi(A)$ on t by Proposition 2. Given an unambiguous n-NSTT A , we trim A , compute its projection $\pi(A)$ and types τ_A in time $O(\mathbf{size}(A) \times n)$. Given a tree $t \in T_\Sigma$ the algorithm proceeds as follows. It enumerates $r \in \mathbf{succ_runs}_A(t)$ with delay $O(\mathbf{size}(A) \times \mathbf{size}(t) \times n)$ per run and return all n -tuples of nodes corresponding to some Boolean annotation τ_A^r .

5 Learning Model and Algorithm

The learning model for words languages from polynomial time and data with positive and negative examples [7,6] can be adapted to tree languages.

Definition 3. *Tree languages over a fixed set Σ represented tree automata in some class \mathcal{C} are called identifiable from polynomial time and data if there exist two polynomials p_1 and p_2 and an algorithm learner such that:*

- for all input samples $S \subseteq T_\Sigma \times \mathbf{Bool}$, learner(S) returns a tree automaton $A \in \mathcal{C}$ in time $O(p_1(|S|))$, that is consistent with S in that for all $t \times b \in S$: $t \in L(A)$ iff $b = 1$;
- for all tree automata $A \in \mathcal{C}$ there exists a so called characteristic sample $\mathbf{char}(A)$ of cardinality less than $p_2(\mathbf{size}(A))$ such that, for all input samples $S \supseteq \mathbf{char}(A)$, learner(S) returns a tree automaton $A' \in \mathcal{C}$ equivalent to A .

In contrast to the case of words, the learning model for trees bounds the *cardinality* of the characteristic sample, not its size. This relaxation may be acceptable as long as one is only interested in the existence of a polynomial time learner. If \mathcal{C} is the class of deterministic tree automata, the learner can be defined by the RPNI algorithm in [17].

The model for learning tree languages is only partially adapted to queries. The question is which examples to use for n -ary queries. Let q be an n -ary query. A *completely annotated example for q* for q is a pair $(t, q(t))$ where $t \in T_\Sigma$. We call t called CARRIER of $(t, q(t))$. For a set S of completely annotated examples for q , we denote by CARRIER(S) the set of supports. A completely annotated example $(t, q(t))$ defines $|q(t)|$ positive examples, i.e. a positive example $(t \times \beta, 1)$ for each tuple tree $t \times \beta$ with β in $q(t)$. It also defines implicit negative examples, i.e. trees $(t \times \beta, 0)$ with β not in $q(t)$ for all t in CARRIER(S).

The cardinality of a completely annotated example $(t, q(t))$ is $q(t) + 1$. The size of a completely annotated example $(t, q(t))$ is $\mathbf{size}(t) + |q(t)| \times n$. A sample

is a set of completely annotated examples for a target query q , its cardinality is the sum of the cardinalities of all completely annotated examples in S , its size is the sum of sizes of all completely annotated examples in S . A tree automaton A over $\Sigma \times \text{Bool}^n$ is *consistent with a sample* S if every tree $t \times \beta$ with $(t, q(t))$ in S and $\beta \in q(t)$ is in $L(A)$ and if there is no tree $t \times \beta$ in $L(A)$ such that $(t, q(t))$ is in S and β is not in $q(t)$.

The model for learning queries is defined w.r.t. a query representation formalism. Two query representations are said to be equivalent if they represent the same query. This leads us to the following definition:

Definition 4. *n -ary queries represented by a query representation formalism \mathcal{R} are said to be identifiable from polynomial time and data from completely annotated examples if there exist two polynomials p_1 and p_2 and an algorithm learner such that:*

- for all input samples S of completely annotated n -ary examples learner(S) returns a representation $A \in \mathcal{R}$ in time $O(p_1(|S|))$ that is consistent with S ;
- for all query representations $A \in \mathcal{R}$ there exists a so called characteristic sample $\text{char}(A)$ for A of cardinality less than $p_2(|A|)$ such that, for all input sample $S \supseteq \text{char}(A)$, learner(S) returns a query representation $A' \in \mathcal{R}$ equivalent to A .

Let us recall that, for a tree t and an n -ary query q , the number of selected n -tuples in $q(t)$ is at most $\text{size}(t)^n$. Therefore, if we consider a target query $q_{n,t}$ that extract all n -tuples of a tree t and no tuple for every other tree, the characteristic sample should contain the completely annotated example $(t, q_{n,t}(t))$ whose cardinality is $\text{size}(t) + \text{size}(t)^n \times n$. This holds for arbitrary query representation formalisms. In order to avoid this blow-up, we restrict ourselves to queries that selects a polynomially bounded number of n -tuples per tree: an n -ary query q over Σ -trees is said to be *polynomially bounded* if there is a polynomial p such that for each trees $t \in T_\Sigma$, $|q(t)| < p(\text{size}(t))$.

Theorem 2. *Polynomially bounded n -ary queries represented by deterministic n -NSTTs are identifiable from polynomial time and data from completely annotated examples.*

Before defining the learning algorithm we recall the basics of the RPNI algorithm for trees. RPNI inputs a sample of positive and negative examples. It first computes an initial deterministic tree automaton which recognizes exactly the set of positive examples in the sample. It then merges states as long as possible while verifying consistency (no negative example in the sample is recognized) and preserving determinism. The order of state fusions matters.

Merging works as follows: Let A be the initial automaton. We consider a partition π of $\text{states}(A)$. The equivalence class of q in that partition is denoted by $\pi(q)$. The quotients of A with respect to π is denoted by A/π . It is the automaton which satisfies $\text{states}(A/\pi) = \pi$ and $\text{final}(A) = \{p \in \pi \mid \pi \cap \text{final}(A) \neq \emptyset\}$. The rules of A are defined such that $(a, \mathbf{b}) \rightarrow q \in \text{rules}(A) \Rightarrow (a, \mathbf{b}) \rightarrow \pi(q) \in$

$\text{rules}(A/\pi)$ and $(f, \mathbf{b})(q_1, q_2) \rightarrow q \in \text{rules}(A), \Rightarrow (f, \mathbf{b})(\pi(q_1), \pi(q_2)) \rightarrow \pi(q) \in \text{rules}(A/\pi)$. State merging is performed by a function $\text{merge}(A, \pi, q_i, q_j)$ that outputs a partition π' such that $\pi'(q_i) = \pi'(q_j)$ and other elements of π are preserved. Function $\text{det-merge}(A, \pi, q_i, q_j)$ first merges q_i and q_j and then performs further merges such that the resulting automaton becomes deterministic.

The learning algorithm `learner` for n -ary queries represented by deterministic n -NSTTs is set to be $\text{RPNI}_{n\text{-NSTT}}$. It is given in Figure 2. It uses the same schema than the RPNI algorithm for tree languages, but with the following features:

- the positive examples are tuple trees $t \times \beta$ for every $t \in \text{CARRIER}(S)$ such that $q(t) \neq \emptyset$ and $\beta \in q(t)$;
- not all deterministic tree automata over $\Sigma \times \text{Bool}^n$ are deterministic n -NSTTs, therefore after every merge we have to check whether the resulting automaton is an n -NSTT, this is done using the τ function (see Proposition 1). Note that, as one never merges states of different type, we denote, for a partition π of $\text{states}(A)$ considered by the algorithm and for a set of states $p \in \pi$, $\tau(p)$ as the type of its states;
- we do not have negative examples, but the hypothesis of completely annotated examples as input allows to define implicit negative examples: $t \times \beta$ such that $(t, q(t)) \in S$ and $\beta \notin q(t)$. As there is a bijection between runs on Σ -trees and answers of a query (see lemma 6), verifying whether an implicit negative example is recognized or not is the same as verifying that the number of runs on the support of the input sample does not grow. This replaces the usual consistency check of RPNI -like algorithms.
- Also, note that RPNI requires an order on states. In the initial automaton, each state can be associated to the single tree that it recognizes; states are then ordered following a fixed order on those trees.

The initial n -NSTT A is consistent with the input sample S because it recognizes exactly the set S^+ of tuple trees constructed from S . Let us suppose that, at every call to det-merge , the n -NSTT A/π is consistent with S . The automaton A/π' satisfies $L(A) \subseteq L(A/\pi')$. To check whether A/π is consistent with S , it is sufficient to test whether there is no new tree $t \times \beta$ in $L(A')$ with $t \in \text{CARRIER}(S)$. From lemma 6, this is equivalent to check whether, for every tree t in $\text{CARRIER}(S)$, the number of successful runs of the projected automaton $\pi(A)$ is equal to $|q(t)|$. Counting the number of successful runs on an input tree can be done in $O(\text{size}(S))$. Note that we do not consider the size of A' because it is lower than the size of A , and the size of A is linear in the size of S .

Also, we compute the τ function described in section 3 on A . As A is an n -NSTT, condition of lemma 4 is satisfied for A . It is easy to verify that those conditions are also satisfied for A/π if and only if there do not exist two states of different types in the same element of π . This is guaranteed by the fact we never merge states of different types.

Thus $\text{RPNI}_{n\text{-NSTT}}$ computes in polynomial time, for every input sample S , an n -NSTT consistent with S . To end the proof of Theorem 2, it remains to prove the second item of Definition 4, i.e. we must define characteristic samples

RPNI_{n-NSTT}

Input: a sample S of completely annotated examples
compute $S^+ = \{t \times \beta \mid t \in \text{CARRIER}(S), \beta \in q(t)\}$
let A be the minimal deterministic n -NSTT such that $L(A) = S^+$
Compute τ and order states of A from q_i to q_n
let $m = \sum_{t \in \text{CARRIER}(S)} |q(t)|$
let π be the trivial partition of $\text{states}(A)$
For $i = 0$ **to** $|\text{states}(A)|$ **do**
 let q be the state with the smallest index in $\pi(q_i)$
 If $q_i = q$ **then** $\%$ q_i has not been merged
 For $j = 0$ **to** $i - 1$ **do**
 If $\tau(q_i) = \tau(q_j)$ **then**
 $\pi' \leftarrow \text{det-merge}(A, \pi, q_i, q_j)$
 let m' be the number of runs of A/π' on $\text{CARRIER}(S)$
 $\%$ test consistency with negative information
 If $m = m'$ **then** $\pi \leftarrow \pi'$ and **Exit Inner Loop**

Output : A/π

Fig. 2. The learning algorithm learner for n -ary queries represented by deterministic n -NSTTs

for n -ary queries represented by deterministic n -NSTTs, and we must prove the convergence property of $\text{RPNI}_{n\text{-NSTT}}$ w.r.t. characteristic samples.

Tree languages represented by deterministic automata are identifiable from polynomial time and data [17]. Thus n -ary queries, considered as tree languages over $\Sigma \times \text{Bool}^n$, represented by deterministic n -NSTTs are identifiable from polynomial time and data. But, recall that this result is true in the learning model from positive and negative examples. Let $\text{learner}' = \text{RPNI}$ be the learning algorithm for tree languages represented by deterministic tree automata and char' be the function computing the characteristic sample associated with a deterministic tree automaton. Let A be a deterministic n -NSTT, $\text{char}'(A)$ is the characteristic sample for A which is the representation of a tree language of $\Sigma \times \text{Bool}^n$ -trees. We define the characteristic sample $\text{char}(A)$ for A which is the representation of an n -ary query by:

$$\text{char}(A) = \{(t, q(t)) \mid (t \times \beta, b) \in \text{char}'(A)\}$$

We show that the cardinality of $\text{char}(A)$ is polynomial. As tree languages represented by deterministic tree automata are learnable from polynomial time and data, there is a polynomial p'_2 such that the cardinality of $\text{char}'(A)$ is less than $p'_2(s)$. Consequently, the number of trees t such that there exists an example $(t \times \beta, b) \in \text{char}'(A)$ is less than $p'_2(s)$. Therefore, $\text{CARRIER}(S)$ has cardinality less than $p'_2(s)$. As we consider polynomially bounded queries, the cardinality of every completely annotated example is polynomial. Thus there is a polynomial p_2 such that the cardinality of $\text{char}(A)$ is less than $p_2(S)$.

Let `learner` be set to $\text{RPNI}_{n\text{-NSTT}}$. We have shown that, for every sample S , $\text{RPNI}_{n\text{-NSTT}}$ outputs in polynomial time an $n\text{-NSTT}$ consistent with S . It remains to show that if $\text{char}(A) \subseteq S$ then $\text{RPNI}_{n\text{-NSTT}}$ with input S outputs an $n\text{-NSTT}$, denoted by $\text{RPNI}_{n\text{-NSTT}}(S)$, equivalent to A .

Let A be the target $n\text{-NSTT}$, let S be a sample that contains $\text{char}(A)$, we define the sample S' of positive and negative examples by:

$$S' = \{(t \times \beta, 1) \mid t \in \text{CARRIER}(S), \beta \in q(t)\} \cup \{(t \times \beta, 0) \mid t \in \text{CARRIER}(S), \beta \notin q(t)\}$$

By definition of $\text{char}(A)$ and of S' , we have $\text{char}'(A) \subseteq S'$. Then, RPNI with input S' outputs a deterministic automaton $\text{RPNI}(S') = A'$ such that $L(A') = L(A)$.

It remains to show that $\text{RPNI}_{n\text{-NSTT}}(S) = \text{RPNI}(S')$. First, verifying that the number of runs on $\text{CARRIER}(S)$ does not grow is equivalent to the consistency test done by RPNI w.r.t. S' (as said above). Second, if $\text{char}(A) \subseteq S$, and consequently $\text{char}'(A) \subseteq S'$, $\text{RPNI}(S') = A'$ is an $n\text{-NSTT}$ because $L(A') = L(A)$ is canonical. Therefore, under the hypothesis that $\text{char}(A) \subseteq S$, at every step of $\text{RPNI}_{n\text{-NSTT}}$, the current deterministic automaton is an $n\text{-NSTT}$. This is because otherwise a tree which is not a tuple tree would be accepted (the sequence of languages is increasing according to inclusion because states are merged). Thus, under the hypothesis that $\text{char}(A) \subseteq S$, merged states will always be of the same type. Thus, $\text{RPNI}_{n\text{-NSTT}}(S) = \text{RPNI}(S')$.

6 $n\text{-NSTT}$ s for Unranked Trees

HTML or XML documents parse into unranked trees where every node may have a list of children of unbounded length, not only two. The notion of $n\text{-ary}$ queries carries over literally.

As an example, consider the unranked tree `film-list` in Fig. 3. This tree represents a list (L) of three films (F), two of which are directed by Hitchcock (H) and one by Wenders (W). The letter (T) represents the title of the film. The binary query `hitchcock` asks for pairs of directors and title in films by Hitchcock. From `film-list`, this query selects the following pairs of nodes: $\text{hitchcock}(\text{film-list}) = \{(1 \cdot 2, 1 \cdot 1), (3 \cdot 2, 3 \cdot 1)\}$. The tree in Fig. 3 is annotated by the first pair $(1 \cdot 2, 1 \cdot 1)$.

For extending $n\text{-NSTT}$ s to unranked trees, we only need a notion of tree automata for unranked trees. It must come with a good notion of bottom-up determinism, for which the Myhill-Nerode theorem holds. This needs some care [11]. We solve this problem as in [1] by using stepwise tree automata [3]. These have the further advantage that they can be identified with standard tree automata operating on binary encodings of unranked trees, so that all our learning algorithms carry over.

An example of stepwise tree automaton inferred by our learning algorithm is given Fig. 3. This automaton has been inferred from completely annotated example for query `hitchcock`, and recognizes that query, at least for documents of the correct type.

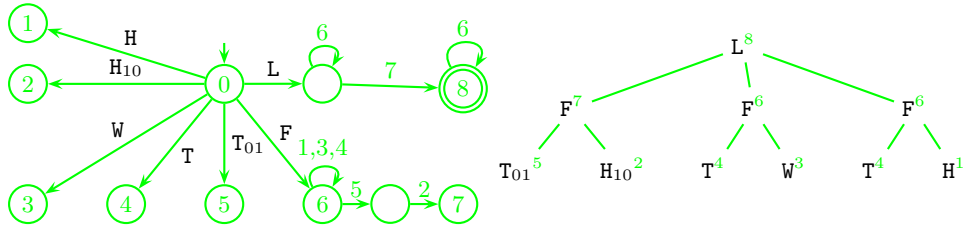


Fig. 3. A stepwise tree automaton inferred by our algorithm $\text{RPNI}_{2\text{-NSTT}}$ (left); the tree film-list annotated by a successful run; state 8 is obtained by evaluating the word $L\cdot 7\cdot 6\cdot 6$. Bit vectors 00 are ignored, so we write L instead of L_{00}

	Okra						Bigbook					
	$\text{RPNI}_{\text{NSTT}}$		$\text{RPNI}_{1\text{-NSTT}}$		$\text{RPNI}_{\text{NSTT}}$		$\text{RPNI}_{1\text{-NSTT}}$					
# Ex.	F-meas.	Init.	infer.	F-meas.	Init.	infer.	F-meas.	Init.	infer.	F-meas.	Init.	infer.
1	100 %	72	24	97.1 %	624	30	68.4 %	162	37	89.4 %	485	29
2	100 %	82	24	98.3 %	547	28	91.3 %	172	42	98.6 %	877	29
3	100 %	85	24	94.3 %	1045	31	100 %	179	48	100 %	1226	30

Fig. 4. Learning monadic queries by RPNI for either NSTTs [2] or 1-NSTTs as proposed here: F-measure, sizes of initial and inferred automata.

7 Application to Web Information Extraction

We have implemented our learning algorithm and started applying it to Web information extraction tasks. We have added a single heuristic proposed in [6], which consists in typing states, so that only trees compatible with HTML syntax are recognized. Textual values and attributes are ignored.

In the case of monadic queries, we compare our algorithm $\text{RPNI}_{1\text{-NSTT}}$ with $\text{RPNI}_{\text{NSTT}}$ from [2]. We use the RISE benchmark: www.isi.edu/info-agents/RISE. Results are averaged over 30 experiments. They are presented in Fig. 4. Our algorithm achieves a little worse on the Okra benchmark, because this benchmark contains pages with a single element to be extracted. On Bigbook, however, $\text{RPNI}_{1\text{-NSTT}}$ performs better than $\text{RPNI}_{\text{NSTT}}$. It is interesting to observe that our technique produce bigger initial automata (because of canonicity, we have one input tree per tuple), but output automata are roughly of the same size for the two systems. These experiments show that induction of NSTTs and 1-NSTTs yield similarly good performance while using different representation schemas.

For n-ary queries, we run $\text{RPNI}_{n\text{-NSTT}}$ on the benchmarks Bigbook and Okra. The results are promising. We also use the Datafoot benchmark available at www.grappa.univ-lille3.fr/~marty/corpus.html. It contains different documents with various structures: lists, tables, rotated tables, cross-tables among others. We learn from only one completely annotated Web document. “Success”

# Examples	Okra			Bigbook		
	F-meas.	Init.	Infer.	F-meas.	Init.	Infer.
1	90.4 %	469	31	89.9 %	505	33
2	97.6 %	781	31	95.2 %	891	33
3	99.4 %	1171	32	100 %	1342	34

Fig. 5. Results of $\text{RPNI}_{2\text{-NSTT}}$ on Okra and Bigbook benchmarks on a binary task: extraction of (name, mail) on Okra and (name, address) on Bigbook.

Dataset	Succ. ?	Description	Dataset	Succ. ?	Description
L0	YES	table with tuples in rows	L5	YES	fake list (sequence of EM)
L1	NO	table with tuples in columns	L6	NO	fake list 2 (sequence of SPAN)
L2	YES	2 column table w/ separator	L7	YES	list of descriptions (DD/DT tag)
L3	YES	nested lists	L8	YES	description and list of SPAN
L4	YES	lists without separator	L9	YES	list of tables, one element factorized

Fig. 6. $\text{RPNI}_{2\text{-NSTT}}$ on Web pages with various structures from the Datafoot benchmark.

means that we achieve 100% F-measure on other web pages. Experimental results are given in Fig. 6. They are generally very positive. Limitation arise only in the case of non regular queries (L1), or when the tree structure alone is not sufficiently informative (L6). These limitations are to be expected of course.

Future Work

Completely annotated examples are not realistic in practice of information extraction. As in the monadic case, we will have to introduce intelligent tree pruning techniques in order to cut of irrelevant parts of documents. This is needed to deal with partially annotated documents, in order to reduce the annotation effort and to improve the quality of inferred queries. It is fundamental to interactive learning of n-ary queries.

References

1. J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducer. *Machine Learning*, 2006.
2. J. Carme, A. Lemay, and J. Niehren. Learning node selecting tree transducer from completely annotated examples. In *ICGI*, vol. 3264 of *LNAI*, p. 91–102. 2004.
3. J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In *RTA*, vol. 3091 of *LNCS*, p. 105 – 118. 2004.
4. B. Chidlovskii. Wrapping web information providers by transducer induction. In *ECML*, vol. 2167 of *LNAI*, p. 61 – 73, 2001.
5. A. Corb , J. Oncina, and P. Garc a. Learning regular languages from a complete sample by error correcting techniques. *IEE*, p. 4/1–4/7, 1993.

6. C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–137, 1997.
7. E.M. Gold. Complexity of automaton identification from given data. *Inf. Cont*, 37:302–320, 1978.
8. G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *17th Annual IEEE Symposium on Logic in Computer Science*, p. 189–202, 2002.
9. H. Hosoya and B. Pierce. Regular expression pattern matching for XML. *Journal of Functional Programming*, 6(13):961–1004, 2003.
10. N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.
11. W. Martens and J. Niehren. On the minimization of XML schemas and tree automata for unranked trees. *Journal of Computer and System Science*, 2006.
12. Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of xpath. *Journal of the ACM*, 51(1):2–45, 2004.
13. I. Muslea, S. Minton, and C. Knoblock. Active learning with strong and weak views: a case study on wrapper induction. In *IJCAI 2003*, p. 415–420, 2003.
14. F. Neven and J. Van Den Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, 49(1):56–100, 2002.
15. J. Niehren, Laurent Planque, J.M. Talbot, and S. Tison. N-ary queries by tree automata. In *DBPL*, vol. 3774 of *LNCS*, p. 217–231. 2005.
16. J. Oncina and P. García. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, p. 49–61, 1992.
17. J. Oncina and P. García. Inference of recognizable tree sets. Tech. report, Universidad de Alicante, 1993. DSIC-II/47/93.
18. S. Raeymaekers, M. Bruynooghe, and J. Van den Bussche. Learning (k,l)-contextual tree languages for information extraction. In *ECML*, vol. 3720 of *LNAI*, p. 305–316, 2005.
19. J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Math. System Theory*, 2:57–82, 1968.