Learning Node Selecting Tree Transducer from Completely Annotated Examples

Julien Carme, Aurélien Lemay, and Joachim Niehren

Mostrare project, INRIA Futurs, Lille, France **

Abstract. A base problem in Web information extraction is to find appropriate queries for informative nodes in trees. We propose to learn queries for nodes in trees automatically from examples. We introduce *node selecting tree transducer* (NSTT) and show how to induce deterministic NSTTs in polynomial time from completely annotated examples. We have implemented learning algorithms for NSTTs, started applying them to Web information extraction, and present first experimental results.

Keywords: Web information extraction, tree automata and logics, grammatical inference.

1 Introduction

Web documents in HTML or XML form trees with nodes containing text. The tree structure is relevant for Web information extraction (IE) from well structured documents as created by databases. Many recent approaches to Web IE therefore focus on tree structure [5, 10, 14] rather than pure text [15, 22].

A base problem in Web information extraction (IE) is to find appropriate queries for informative nodes in trees. In Fig. 1, for instance, one might want to extract all Email addresses (single-slot IE). This can be done by querying for all links in the last line of each table, i.e., for all A-nodes whose TR-node ancestor is the last child of some TABLE-node. Alternatively, one might want to ask for all pairs of names and email addresses (multi-slots IE). This problem can be reduced to iterated single-slot IE: first search for all tables that encompass the pairs and then extract the components from the tables.

Gottlob et. al. [10] advocate for monadic Datalog as representation language for node queries in trees. This logic programming language is highly expressive (all regular node queries in trees can be expressed) while enjoying efficient algorithms for answering queries. The Lixto system for multi-slot IE [1] supplies a graphical user interface by which to interactively specify and test node queries in monadic Datalog.

Tree automata yield an alternative representation formalism [18, 21, 8, 3] that is particularly relevant for grammatical inference. Run-based node queries by

** This research was partially supported by: "CPER 2000-2006, Contrat de Plan état - région Nord/Pas-de-Calais: axe TACT, projet TIC"; fonds européens FEDER "TIC - Fouille Intelligente de données - Traitement Intelligent des Connaissance" OBJ 2-phasing out - 2001/3 - 4.1 - n 3. And by "ACI masse de données ACIMDD"

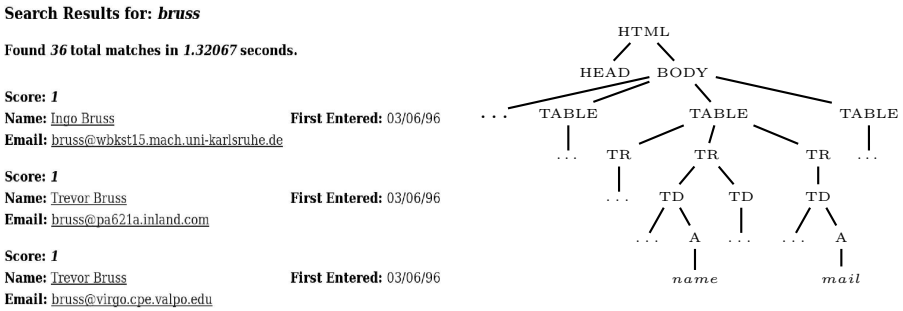

Fig. 1. A simple Web page, its corresponding tree.

tree automata can be translated in linear time into monadic Datalog, and in non-elementary time back due to Thatcher and Wright's famous theorem [23]. A more recent problem here is to deal with the *unrankedness* of trees [3] as in `HTML` and `XML`, by encoding into binary trees.

The objective of the present paper is to learn tree automata that represent node queries in trees from completely annotated examples. Query induction might be useful in order to circumvent manual query specification as in Lixto. Completely annotated examples consist of trees where all nodes are annotated by Booleans, stating whether a node is selected or not. We introduce *node selecting tree transducers* (`NSTT`) for representing node queries in trees, and show how to infer *deterministic* `NSTT`s from examples by variants of the `RPNI` algorithm [19]. We have implemented several versions of our learning algorithm and started to apply them to Web IE, so that we can present first experimental results.

Related Work. Kosala et. al. [14] learn tree automata representing node queries in trees from less informative examples, which specify selected nodes but not unselected ones. This has the advantage that complete annotations for all nodes are not needed, but restricts the class of learnable queries to those representable by local or k-testable tree automata. Deterministic `NSTT`s, in contrast, can represent all regular node queries in trees (see subsequent work [2]).

Node queries of bounded length in monadic-second order (MSO) logic over trees are shown `PAC` learnable in [11]. Variants of `RPNI` for inducing sub-sequential text transducers were proposed in [12]; these transducers may alter the structure of words in contrast to `NSTT`s which only relabel nodes in trees. Chidlovskii [4] proposes induction of word transducers for Web IE.

2 Node Queries in Binary Trees

Before considering the particularities of `HTML` and `XML` trees, we will deal with node queries for binary trees. We start from an finite alphabet Σ consisting of binary function symbols f and constants a. A *binary tree* t over Σ is a term that satisfies the grammar:

$$t ::= f(t_1, t_2) \mid a$$

For every tree t let $\mathsf{nodes}(t) \subseteq \{1,2\}^*$ be the set of *nodes* of tree t. This set determines the shape t, i.e., two trees have the same shape if and only if they have the same node sets. The empty word ϵ always addresses the root of a tree; the first child of node v is node $v1$ and its second child is $v2$. We write $t(v)$ for the label of nodes $v \in \mathsf{nodes}(t)$. The size $|t|$ is the cardinality of $\mathsf{nodes}(t)$. The tree $t = f_1(a_1, f_2(a_1, a_2))$, for instance, has the node set $\{\varepsilon, 1, 2, 21, 22\}$. Its root is labeled by $t(\varepsilon) = f_1$ while its third leaf is labeled by $t(22) = a_2$.

Definition 1. *A (monadic)* node query in binary trees *is a function q that associates to each binary tree t a set of nodes $q(t) \subseteq \mathsf{nodes}(t)$.*

The query **leaf** associates the set of leaves to a given tree, i.e., those nodes that don't have children. If $t = f(a_1, f(a_1, a_2))$ then $\mathbf{leaf}(t) = \{11, 21, 22\}$.

3 Tree Automata

We are interested in regular node queries in trees that can be defined equivalently by tree automata, in monadic Datalog, or MSO over trees. (See [3] for the case of unranked trees.) Here, we recall a representation formalisms for node queries based on successful runs of tree automata [21, 8, 3].

A *tree automaton* A over signature Σ consists of a finite set $\mathsf{states}(A)$ of states, a set $\mathsf{final}(A) \subseteq \mathsf{states}(A)$ of final states, and a finite set $\mathsf{rules}(A)$ of rules of the form $f(p_1, p_2) \to p$ or $a \to p$ where $p, p_1, p_2 \in \mathsf{states}(A)$, f a binary function symbol and a a constant in Σ. The *size* $|A|$ of a tree automaton A is the number of its states plus the number of symbols occurring in its rules.

A *run* r of a tree automaton A on a tree t is a binary tree over $\mathsf{states}(A)$ of the same shape, i.e. $\mathsf{nodes}(r) = \mathsf{nodes}(t)$, such that for all $v \in \mathsf{nodes}(t)$ and $f, a \in \Sigma$: if $t(v) = f$ then $f(r(v1), r(v2)) \to r(v) \in \mathsf{rules}(A)$, and if $t(v) = a$ then $a \to r(v) \in \mathsf{rules}(A)$. A run r of A on t is *successful* if $r(\varepsilon) \in \mathsf{final}(A)$.

Let A_0 be an automaton with 3 states $0, 1, 2$, a single final state 2, and the rules: $a \to 0,\ a \to 1,\ f(0,1) \to 2$. The tree $f(a,a)$ has a unique successful run by A_0, the tree $2(0,1)$; no other tree permits a successful run with A_0.

Definition 2. *A tree automaton is (bottom-up)* deterministic *if no two rules have the same left-hand side and* unambiguous *if no tree permits two successful runs.*

Every deterministic tree automaton is clearly unambiguous, but not conversely. The automaton A_0 above yields a counter example. It is nondeterministic given that tree a permits two distinct runs with A_0, but nevertheless unambiguous given that none of these two runs is successful.

We write $\mathsf{runs}_A(t)$ for the set of all runs of automaton A on tree t and $\mathsf{succ\_runs}_A(t)$ for the subset of all successful runs. A tree automaton *recognizes* all trees t that permit a successful run of A on t. The *language* $L(A)$ of an automaton A contains all trees that A recognizes.

A pair of a tree automaton A over Σ and a set of selection states $P \subseteq \mathsf{states}(A)$ defines a node query in trees $t \in \mathsf{tree}_\Sigma$:

$$\mathsf{query}_{A,P}(t) = \{v \mid r \in \mathsf{succ\_runs}_A(t), r(v) \in P\}$$

We call a monadic query q *regular* if it is equal to some $\mathsf{query}_{A,P}$. Thatcher and Wright's famous theorem [23] proves that a query is regular if and only if it can be defined by a MSO-formula over binary trees with a single free node variable.

4 Node Selection Tree Transducer

We will introduce NSTTs, a new representation formalism for regular node queries in trees suitable for grammatical inference.

Every subset $q(t) \subseteq \mathsf{nodes}(t)$ can be identified with a tree over Bool with the same shape, such that the Boolean values $q(t)(v)$ satisfy for all $v \in \mathsf{nodes}(t)$:

$$q(t)(v) \leftrightarrow v \in q(t)$$

The query application $\mathbf{leaf}(f(a, g(a, b))$, for instance, yields the Boolean tree $\mathsf{false}(\mathsf{true}, \mathsf{false}(\mathsf{true}, \mathsf{true}))$.

Given two trees t over Σ and β over Bool with the same shape, we define a tree $t \times \beta$ over $\Sigma \times \mathsf{Bool}$ by requiring for all $v \in \mathsf{nodes}(t) = \mathsf{nodes}(\beta) = \mathsf{nodes}(t \times \beta)$:

$$(t \times \beta)(v) = (t(v), \beta(v))$$

A tree language L over $\Sigma \times \mathsf{Bool}$ is *functional* (resp. *total*) if for all $t \in \mathsf{tree}_\Sigma$ there exists at most (resp. at least) one tree $\beta \in \mathsf{tree}_{\mathsf{Bool}}$ such that $t \times \beta \in L$.

We associate tree languages lan_q over $\Sigma \times \mathsf{Bool}$ to queries q in trees over Σ:

$$\mathsf{lan}_q = \{t \times q(t) \mid t \in \mathsf{tree}_\Sigma\}$$

Tree languages lan_q are always functional and total. Conversely, we will associate queries to functional tree languages. Let $\mathsf{total}(L)$ be the functional total language containing L and contained in $L \cup \mathsf{tree}_{\Sigma \times \{\mathsf{false}\}}$. Functional languages L define node queries in trees that satisfies for all trees $t \in \mathsf{tree}_\Sigma$ and $\beta \in \mathsf{tree}_{\mathsf{Bool}}$:

$$\mathsf{query}_L(t) = \beta \text{ iff } t \times \beta \in \mathsf{total}(L)$$

We have $\mathsf{query}_{\mathsf{lan}_q} = q$ for all node queries q in trees and $\mathsf{lan}_{\mathsf{query}_L} = \mathsf{total}(L)$ for all functional tree languages L. For every functional tree language L there exists exactly one query q with $\mathsf{lan}_q = L$, but for some queries q there exist many functional tree languages L with $\mathsf{total}(L) = \mathsf{lan}_q$. This ambiguity has to be treated carefully.

Definition 3. *An* NSTT *is a tree automaton A whose language $L(A)$ is functional. A* NSTT-query *has the form $\mathsf{query}_{L(A)}$ where A is an NSTT.*

Proposition 1. *NSTT-queries are regular.*

Proof. Given a NSTT A we define a projection tree automaton $\pi(A)$ over Σ. We set $\mathsf{states}(\pi(A)) = \mathsf{states}(A) \times \mathsf{Bool}$, $\mathsf{final}(\pi(A)) = \mathsf{final}(A) \times \mathsf{Bool}$, and fix the following automata rules by:

$$\frac{(f,b)(p_1,p_2) \to p \in \mathsf{rules}(A)}{f((p_1,b_1),(p_2,b_2)) \to (p,b) \in \mathsf{rules}(\pi(A))} \qquad \frac{(a,b)(p_1,p_2) \to p \in \mathsf{rules}(A)}{a \to (p,b) \in \mathsf{rules}(\pi(A))}$$

The subsequent Lemma 1 yields $\mathsf{query}_{L(A)} = \mathsf{query}_{\pi(A),\mathsf{states}(A)\times\{\mathsf{true}\}}$.

Lemma 1. *$r \in \mathsf{runs}_A(t \times \beta)$ iff $r \times \beta \in \mathsf{runs}_{\pi(A)}(t)$.*

The converse of Prop. 1 holds as proved by the authors a follow up paper [2]. The construction in the proof shows NSTT-queries can be answered efficiently, in that $\mathsf{query}_{L(A)}(t)$ can be computed in time $O(|A| * |t|)$ from NSTTs A and trees t. It is sufficient to transform NSTT-queries into automata queries (in linear time) which can be answered in linear time. We finally present a new polynomial time algorithm for testing whether a deterministic automaton A is an NSTT.

Proposition 2. *The language of a deterministic tree automaton A over $\Sigma \times \mathsf{Bool}$ is functional if and only if the projection $\pi(A)$ to Σ is unambiguous.*

Proof. For the one direction, let A be deterministic (and thus unambiguous) and $L(A)$ functional. Suppose $r_1 \times \beta_1, r_2 \times \beta_2 \in \mathsf{succ\_runs}_{\pi(A)}(t)$ for some $t \in \mathsf{tree}_\Sigma$. Lemma 1 yields $r_1 \in \mathsf{succ\_runs}_A(t \times \beta_1)$ and $r_2 \in \mathsf{succ\_runs}_A(t \times \beta_2)$. The functionality of A implies that $t \times \beta_1 = t \times \beta_2$ and thus $\beta_1 = \beta_2$. The unambiguity of A yields $r_1 = r_2$ so that $r_1 \times \beta_1 = r_2 \times \beta_2$. This proves that $\pi(A)$ is unambiguous.

For the converse, assume that $\pi(A)$ is unambiguous and suppose $t \times \beta_1, t \times \beta_2 \in L(A)$. Let $r_1 \in \mathsf{succ\_runs}_A(t \times \beta_1)$ and $r_2 \in \mathsf{succ\_runs}_A(t \times \beta_2)$. Lemma 1 yields $r_1 \times \beta_1, r_2 \times \beta_2 \in \mathsf{succ\_runs}_{\pi(A)}(t)$. The unambiguity of $\pi(A)$ implies $\beta_1 = \beta_2$, i.e., $L(A)$ is functional.

Proposition 3. *Unambiguity of tree automata can be tested in polynomial time.*

Proof. An algorithm for word automata can be found in [6]. Here, we test unambiguity for tree automata A. The algorithm is based on the binary relation $\mathsf{drst}_\mathsf{A} \subseteq \mathsf{states}(A) \times \mathsf{states}(A)$, where $\mathsf{drst}_\mathsf{A}(p,p')$ means that two distinct runs of A label the root of the same tree with respectively p and p'.

$$\mathsf{drst}_\mathsf{A}(p,p') \text{ iff } \exists t \in \mathsf{tree}_\Sigma \exists r, r' \in \mathsf{runs}_A(t).\ r \neq r' \wedge r(\epsilon) = p \wedge r'(\epsilon) = p'$$

The automaton A is ambiguous if and only if there exists $p \in \mathsf{final}(A)$ such that $\mathsf{drst}_\mathsf{A}(p,p)$. It remains to compute the relation drst_A. We assume that A does not contain useless states, which are not used in any runs. We compute the relation by applying the rules of Fig. 2 exhaustively.

The first rule permits to derive runs for constant a-trees leading into distinct states. If we can apply the second rule, there are trees t_1, t_2 with runs into states p_1, p_2 respectively, since there are no useless states. The tree $f(t_1,t_2)$ then

$$\mathsf{drst}_A(p,p') \Leftrightarrow$$
$$\begin{array}{l}
\quad (a \to p \in \mathsf{rules}(A) \wedge a \to p' \in \mathsf{rules}(A) \wedge p \neq p') \\
\vee\ (f(p_1,p_2) \to p \in \mathsf{rules}(A) \wedge f(p_1,p_2) \to p' \in \mathsf{rules}(A) \wedge p \neq p') \\
\vee\ (f(p_1,p_2) \to p \in \mathsf{rules}(A) \wedge f(p'_1,p_2) \to p' \in \mathsf{rules}(A) \wedge \mathsf{drst}_A(p_1,p'_1)) \\
\vee\ (f(p_1,p_2) \to p \in \mathsf{rules}(A) \wedge f(p_1,p'_2) \to p' \in \mathsf{rules}(A) \wedge \mathsf{drst}_A(p_2,p'_2)) \\
\vee\ (f(p_1,p_2) \to p \in \mathsf{rules}(A) \wedge f(p'_1,p'_2) \to p' \in \mathsf{rules}(A) \wedge \mathsf{drst}_A(p_1,p'_1) \wedge \mathsf{drst}_A(p_2,p'_2))
\end{array}$$

Fig. 2. Testing non-ambiguity

permits two distinct runs in p'_1 and p'_2. The third rule is recursive. Suppose, there are distinct runs on t_1 leading to p_1 and p'_1 and a run on t_2 into p_2, then there exists distinct runs on $f(t_1,t_2)$ leading into p and p' respectively (even if $p = p'$). The forth and fifth rule are similar.

The whole algorithm is polynomial in the size of A. We can apply at most $\mathsf{states}(A)^2$ rules, when avoid to infer the same pair $\mathsf{drst}_A(p,p')$ twice. Every rule application can be done in polynomial time in the size of the automaton.

Corollary 1. *Testing whether a deterministic tree automaton over $\Sigma \times \mathsf{Bool}$ is an NSTT can be done in polynomial time.*

Proof. For a deterministic NSTT A, we check the non-ambiguity of its projection $\pi(A)$ (Prop. 2). This can be done in polynomial time (Prop. 3).

5 Inducing NSTT-Queries

We show how to induce NSTT-queries for nodes in trees from completely annotated examples, i.e., pair trees of the form $t \times \beta$. Let $\mathsf{samples}^{\mathsf{a}}$ be the set of all completely annotated examples. Every tree $t \times q(t)$ is a completely annotated example for q. Complete annotations thus specify for all nodes of a tree, whether they are selected or not. Given that completely annotated examples express positive and negative information, it should not come as a surprise that we will learn by the RPNI algorithm for regular positive and negative inference [16, 19, 20].

5.1 Identification in the Limit

We recall the learning model of identification in the limit [9] and apply it to identification of node queries in trees.

Definition 4. *Let* class *and* examples *be sets related by a binary relation called consistency, and $\equiv$ an equivalence relation on* class. *Let* samples *be the set of all finite subset of* examples. *A sample in* samples *is consistent with a class member if all its examples are.*

Members of class *are* identifiable in the limit *from* examples *if there are computable functions* learner : samples $\to$ class *mapping samples to class members and* char : class $\to$ samples *computing consistent samples for all class members – called characteristic samples – such that* $\mathsf{learner}(S) \equiv M$ *for every member* $M \in \mathsf{class}$ *and sample* $S \supseteq \mathsf{char}(M)$ *consistent with* M.

Identification of NSTT-queries from completely annotated examples is an instance of Def. 4 where $\mathsf{class} = \{\mathsf{query}_{L(A)} \mid A \text{ is an } \mathtt{NSTT}\}$, $\mathsf{examples}$ contains all completely annotated examples, and equivalence $\equiv$ of NSTT-queries is equality. An annotated example $t \times \beta$ is consistent with a query q if $q(t) = \beta$.

Identifying deterministic NSTTs from completely annotated samples is another case. Here $\mathsf{class} = \{A \mid A \text{ is an } \mathtt{NSTT}\}$, and an NSTT A is consistent with a completely annotated example if $\mathsf{query}_{L(A)}$ is. The equivalence $A \equiv A'$ is total language equality $\mathsf{total}(L(A)) = \mathsf{total}(L(A'))$.

Identification of regular tree languages from positive and negative examples is the third case. Given some set $T = T_\Sigma$ of trees, let $\mathsf{class} \subseteq 2^{2^T}$ be a class of regular tree languages. A *positive example* is an element of $T \times \{\mathsf{true}\}$, a *negative example* an element of $T \times \{\mathsf{false}\}$. Let $\mathsf{samples}(T)$ be the set all samples with positive or a negative example. An example (t, b) is consistent with a language $L \in \mathsf{class}$ if $t{\in}L \Leftrightarrow b$. Equivalence $\equiv$ of tree languages is language equality.

Identification of deterministic tree automata from positive and negative examples is similar. Again, two automata are equivalent if they have the same language; an automaton is consistent with an examples if its language is.

Proposition 4. *Identification of NSTT-queries from completely annotated examples can be reduced to identification of regular tree languages from positive and negative examples.*

Proof. Completely annotated examples $t \times \beta$ for a query q correspond positive examples $(t \times \beta, \mathsf{true})$ for lan_q and furthermore imply negative example $(t \times \beta', \mathsf{false})$ for all $\beta' \neq \beta$. Let $T = \mathsf{tree}_{\Sigma \times \mathsf{Bool}}$. We define a function $\mathsf{pn} : \mathsf{samples}^{\mathsf{a}} \to \mathsf{samples}(T)$ by samples with completely annotated examples to samples of positive and negative examples:

$$\mathsf{pn}(S) = \{(t \times \beta', b) \mid t \times \beta \in S, b \Leftrightarrow (\beta = \beta')\}$$

Let class the class of regular tree languages over T. We assume that languages in class can be identified from positive and negative examples by functions $\mathsf{learner} : \mathsf{samples}(T) \to \mathsf{class}$ and $\mathsf{char} : \mathsf{class} \to \mathsf{samples}(T)$.

Let class' be the set of NSTT-queries. We show that queries in class' can be identified from completely annotated example by the following functions:

$$\mathsf{learner}' : \mathsf{samples}^{\mathsf{a}} \to \mathsf{class}',\ \mathsf{learner}'(S) = \mathsf{query}_{\mathsf{learner}(\mathsf{pn}(S))}$$
$$\mathsf{char}' : \mathsf{class}' \to \mathsf{samples}^{\mathsf{a}},\quad \mathsf{char}'(q) = \{t \times \beta \in \mathsf{lan}_q \mid (t \times \beta', b) \in \mathsf{char}(\mathsf{lan}_q)\}$$

First note that $\mathsf{char}'(q)$ is always consistent with q. If $t \times \beta \in \mathsf{char}'(q)$ then by definition $t \times \beta \in \mathsf{lan}_q$ and thus $q(t) = \beta$.

Second, notice that $\mathsf{pn}(\mathsf{char}'(q)) \supseteq \mathsf{char}(\mathsf{lan}_q)$. For positive examples $t \times \beta' \in \mathsf{char}(\mathsf{lan}_q)$, clearly $t \times \beta' \in \mathsf{char}'(q)$. For negative examples $(t \times \beta', \mathsf{false}) \in \mathsf{char}(\mathsf{lan}_q)$ there is some $t \times \beta \in \mathsf{lan}_q$ since lan_q is total, with $\beta \neq \beta'$ since lan_q is functional and $\mathsf{char}(\mathsf{lan}_q)$ consistent with lan_q. Hence $t \times \beta \in \mathsf{char}'(q)$ so that $(t \times \beta', \mathsf{false}) \in \mathsf{pn}(\mathsf{char}'(q))$.

Let $q \in \mathsf{class}$ and $S \supseteq \mathsf{char}'(q)$ consistent with L. Then $\mathsf{pn}(S) \supseteq \mathsf{char}(q)$ and $\mathsf{pn}(S)$ consistent with lan_q. Identifying regular languages yields $\mathsf{learner}(\mathsf{pn}(S)) = \mathsf{lan}_q$, i.e. $\mathsf{learner}'(S) = \mathsf{query}_{\mathsf{lan}_q} = q$ as required for identifying NSTT-queries.

Theorem 1. *NSTT-queries for nodes in trees can be identified in the limit from completely annotated examples.*

Proof. The problem is equivalent to identifying regular tree languages from positive and negative examples (Prop. 4) which can be done by `RPNI` [16, 19, 20].

5.2 Identification in Polynomial Time and Data

We now show how to identify deterministic `NSTT`s in polynomial time from completely annotated examples.

Identification in polynomial time and data is identification in the limit with functions learner in polynomial time and char in polynomial space [7].

The classical `RPNI`-algorithm identifies deterministic automata from positive and negative examples in polynomial time and data [19, 16]. It applies to tree languages similarly as to word languages [20]. For a given signature, it computes a polynomial time function $\mathsf{learner}_{\texttt{RPNI}}$ mapping samples of positive and negative examples to deterministic tree automata, and polynomial space function $\mathsf{char}_{\texttt{RPNI}}$ of the inverse type.

Theorem 2. *Let a completely annotated example $t \times \beta$ be consistent with a NSTT A if $\mathsf{query}_{L(A)}(t) = \beta$, and two NSTTs be equivalent if they define the same queries. Given this consistency and equivalence relations, we can identify deterministic NSTTs in polynomial time and data from completely annotated examples.*

Proof. For identifying deterministic `NSTT`s from completely annotated examples, we want to compute the function $\mathsf{learner}_{\texttt{RPNI}} \circ \mathsf{pn}$ which inputs completely annotated examples over $\Sigma \times \mathsf{Bool}$, transforms them into positive and negative examples and then outputs the result of the `RPNI` learner for the signature $\Sigma \times \mathsf{Bool}$. This output is a `NSTT` whose associated language is total, which is the representant of its equivalence class.

Unfortunately, the function pn is not in polynomial space, given that the size of its output may be exponential. In order to solve this problem, we propose a more efficient implementation of the function $\mathsf{learner}_{\texttt{RPNI}} \circ \mathsf{pn}$, the algorithm `tRPNI`, a variant of `RPNI`.

Lets us recall the `RPNI`-algorithm [20, 19, 16]. `RPNI` inputs a sample of positive and negative examples. It first computes a deterministic automaton which recognizes the set of positive examples in the sample. It then merges states exhaustively in some fixed order. A merging operation applies to the recent deterministic automaton A and two states $q_1, q_2 \in \mathsf{states}(q)$ and returns a deterministic automaton $\mathsf{det\_merge}(A, q_1, q_2)$. A deterministic merge is a merge followed by recursive merges needed to preserve preserve determinism. For example, merging q_1 and q_2 in an automaton with rules $f(q_1) \to q_3$ and $f(q_2) \to q_4$ requires merging q_3 with q_4. A merging operation is licensed only if $\mathsf{det\_merge}(A, q_1, q_2)$ is consistent with all negative examples in the sample. The main loop of `RPNI` thus performs at most quadratically many functionality tests and merging operations.

The algorithm `tRPNI` behaves as `RPNI` except that it checks differently whether deterministic merging operation are licenced. It tests whether the language of

$(a, \mathsf{true}) \to 2$
$(a, \mathsf{false}) \to 1$
$(@, \mathsf{false})(2, 2) \to 3$
$(@, \mathsf{false})(3, 1) \to 4$
$(@, \mathsf{false})(2, 4) \to 5$
Final states : $\{5\}$

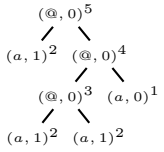

Fig. 3. The initial automaton and its run on the example tree

$(a, \mathsf{true}) \to 2$
$(a, \mathsf{false}) \to 1$
$(@, \mathsf{false})(2, 2) \to 1$
$(@, \mathsf{false})(1, 1) \to 2$
Final states : $\{1\}$

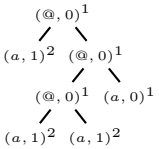

Fig. 4. The inferred NSTT and its run on the example tree

$\mathsf{det\_merge}(A, q_1, q_2)$ is functional. It thereby avoids to enumerate implicit negative examples for functional language once and for all. And fortunately, we can check for functionality in polynomial time in the automaton size (Corollary 1).

Conversely, we compute characteristic samples for NSTTs A as we did before for $\mathsf{query}_{L(A)}$ in the proof of Prop. 4: $\mathsf{char}(A) = \{t \times \beta \in \mathsf{total}(L(A)) \mid (t \times \beta', b) \in \mathsf{char}_{\mathtt{RPNI}}(A)\}$. Since $\mathsf{lan}_{\mathsf{query}_{L(A)}} = \mathsf{total}(L(A))$ it follows that all examples in $\mathsf{char}(A)$ are consistent with A or with ony other NSTT in its equivalence class.

5.3 Example

We illustrate how the learning algorithm works in a simplified case. We want to extract leaves that are on an odd level on trees on the alphabet $\Sigma = \{@, a\}$.

The first step of the algorithm is the construction of the initial automaton. This automaton, and its run on the input sample is indicate in Fig. 3. Merges are then being performed following the order of states. $\mathsf{det\_merge}(A_2, 1, 2)$ is rejected for lack of functionality. Following merge, $\mathsf{det\_merge}(A_3, 1, 3)$, is then accepted. Then $\mathsf{det\_merge}(A_4, 1, 4)$ is rejected. $\mathsf{det\_merge}(A_4, 2, 4)$ is accepted (which implies the merges of states 5 and 1 by propagation of the determinism). This results in the automaton presented in Fig. 4. The example being well chosen, it appears that this NSTT performs the wanted annotation.

6 Application to Web Information Extraction

We have implemented the tRPNI algorithm and started applying it to Web IE. We report first results of this work in progress and discuss some of the problems that arise.

6.1 Modeling HTML Trees

Unranked trees. HTML or XML form *unranked trees*. We encode unranked trees into binary trees. We use the Currying inspired encoding of [3] in order to map to stepwise tree automata, rather than the more frequent first-child next-sibling encoding as in selection automata [8].

The unranked tree TABLE(TR(TD),TR(TD),TR(TD)), for instance, is translated into the binary tree TABLE@(TR@TD)@(TR@TD)@(TR@TD) with a single binary symbol @. Completely annotated examples for unranked trees are translated

into completely annotated examples on ranked encodings, too. Node annotations in unranked trees become leaf annotations in binary encodings. Inner `@`-nodes in binary encodings correspond to edges in unranked trees. As we do not wish to select edges, we label all `@`-nodes by false.
Infinite alphabets. Leaves of `HTML` trees may contain arbitrary texts. So far however, we assumed finite alphabets. In our experiments, we ignore leaf contents completely, by abstracting all texts into a single symbol.
Node attributes. Node attributes are currently ignored. Each `HTML` tag is abstracted into its corresponding symbol, whatever its attributes are.

6.2 Approaching Problems

A nice feature of `RPNI` is that does never do wrong generalizations when applied to a characteristic sample. In practice, however, we only dispose small samples that are seldom characteristic.
Wrong generalizations. Lacking negative information is particularly embarrassing, as it leads to wrong generalizations. This may have the consequence that parts of documents cannot be recognized by inferred `NSTT`s, so that `NSTT`-queries do not select any nodes from such documents. We have designed two heuristics to deal with that problem.
Typed merging. We use typing as inspired from [13] and forbid states with the different types to be merged. So far, we experiment with a fairly basic typing system: leaves of binary encodings are typed by their corresponding `HTML` tag, while inner nodes inherit the type of their first child. Types in annotated examples become types in the initial automaton. This prevents many wrong generalization, while allowing most of the meaningful ones. Our typing reflect the structure of encodings of unranked trees and that we do not want to merge nodes with different `HTML` tags.
Wild-card interpretation. We relax the querying interpretation of inferred `NSTT`s. Consider a tree $t_1@t_2$. If our deterministic `NSTT` does not have any run on t_2, but a run for t_1 leading into state q_1 and if there exists a single rule of the form $q_1@q_2 \to q$ then we permit relaxed runs of $t_1@t_2$ into q that labels all nodes in t_2 by wild-cards. Nodes labeled by wild-cards are never selected.

6.3 Experiments

We have experimented with Okra and Bigbook tasks from the RISE benchmarks(`www.isi.edu/info-agents/RISE`). Both Okra and Bigbook are computer generated web pages which represent sets of personal informations. The extraction task is to extract e-mail addresses. Results are given Fig.5. In Okra, one example is enough to achieve good results. Without wild-cards, performances are weaker in Bigbook, even though the task does not seem more complex than with Okra. Bigbook illustrates the problem of wrong generalizations. At the end of each document, every letter of the alphabet is indicated, either as a link or as standard text. With few examples, `tRPNI` fails to infer a `NSTT` that recognizes

Okra				
# of Examples	Accuracy	size of initial NSTT	size of inferred NSTT	Learning time
1	100 %	72	24	1.02 s
2	100 %	82	24	1.24 s
3	100 %	85	24	1.34 s
Bigbook				
1	76% / 100 %*	162	37	4.14 s
2	85% / 100 %*	172	42	7.12 s
3	100 %	179	48	9.52 s

Fig. 5. Experimental results on Okra and Bigbook benchmarks. (* indicates results without and with the use of wild-cards).

this part of the document, which is totally irrelevant to the querying task. At the same time, other relevant part of the document are properly labelled. Permitting wild-card interpretations helps in this case.

Experiments with more complex tasks so far often yield poor results, because of lack of informations in pure structure and wrong generalizations.

6.4 Possible Improvements

Better text abstraction. The conversion of `HTML` node information into symbols could be improved using text-based information extraction techniques. For instance, instead of using one generic symbol for leaves, one could classify leafs in several clusters such as `names`, `dates`, `numbers`, etc.
Better merging orders. A crucial parameter of `RPNI` (and `tRPNI`) is the order in which state merges are performed. The technique of *evidence driven state merging* [17] could be applied here. The distance between nodes in the encoded unranked trees should be taken into account in this order.

7 Conclusion

We have presented node selecting tree transducer to represent node queries in trees. We have proposed a variant of `RPNI` that can identify `NSTT`s in polynomial time from annotated examples. We have started applying our learning algorithm to Web information extraction and could report first encouraging results.

In follow up work [3], we have shown that all regular queries in trees can be represented by `NSTT`s. On open theoretical question is, whether deterministic `NSTT`s with a total languages can be identified in polynomial time.

In future work, we plan to continue improving our learning algorithms in practice. The open challenge remains, to built feasible and reliable learning based systems for Web information extraction.

References

1. R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *The VLDB Journal*, pages 119–128, 2001.

2. J. Carme, A. Lemay, J. Niehren, and A. Terlutte. Learning regular node queries in trees from completely annotated examples. Forthcoming. 2004.
3. J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. Intenational Conf. on Rewriting Techniques and Applications, 2004.
4. B. Chidlovskii. Wrapping web information providers by transducer induction. In *Proc. ECML*, volume 2167 of *LNAI*, p 61 – 73, 2001.
5. W. W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Web Document Analysis: Challenges and Opportunities*, MPAI, WSP 2003.
6. F. Coste and D. Fredouille. Efficient ambiguity detection in c-nfa. In *Grammatical Inference: Algorithms and Applications*, *LNAI 1891*, 2000.
7. C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–137, 1997.
8. M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees. In *Proc. LICS 2003*, 2003.
9. E. Gold. Language identification in the limit. *Inform. Control*, 10:447–474, 1967.
10. G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *LICS*, 2002.
11. M. Gruhe and G. Turan. Learnability and definability in trees and similar structures. In *STACS'02*, *LNCS 2285*, p 645–658. 2002.
12. P. Garcia. J. Oncina and E. Vidal. Learning subsequential transducers for pattern recognition and interpretation tasks. *IEEE Trans. Patt. Anal. and Mach. Intell.*, 15:448–458, 1993.
13. C. Kermorvant and C. de la Higuera. Learning language with help. In *ICGI 2002*.
14. R. Kosala, M. Bruynooghe, J. V. den Bussche, and H. Blockeel. Information extraction from web documents based on local unranked tree automaton inference. In *(IJCAI-2003)*.
15. N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, Univ. of Washington, 1997.
16. K. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *Workshop on Comput. Learning Theory*, p 45–52. ACM Press, 1992.
17. K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *ICGI 98*, *LNAI 1433*.
18. F. Neven and T. Schwentick. Query automata over finite trees. *TCS*, 275(1-2):633–674, 2002.
19. J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, p 49–61, 1992.
20. J. Oncina and P. García. Inference of recognizable tree sets. Tech. report, Universidad de Alicante, 1993. DSIC-II/47/93.
21. H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *Proceedings of the Symposium on Principles Of Database Systems*, p 155–166, 2003.
22. S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.
23. J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 1968.