Mach Learn (2007) 66:33–67
DOI 10.1007/s10994-006-9613-8

# Interactive learning of node selecting tree transducer

**Julien Carme · Rémi Gilleron · Aurélien Lemay · Joachim Niehren**

Received: 4 July 2005 / Revised: 12 May 2006 / Accepted: 22 June 2006 / Published online: 16 August 2006


**Abstract** We develop new algorithms for learning monadic node selection queries in unranked trees from annotated examples, and apply them to visually interactive Web information extraction.

We propose to represent monadic queries by bottom-up deterministic Node Selecting Tree Transducers (NSTTs), a particular class of tree automata that we introduce. We prove that deterministic NSTTs capture the class of queries definable in monadic second order logic (MSO) in trees, which Gottlob and Koch (2002) argue to have the right expressiveness for Web information extraction, and prove that monadic queries defined by NSTTs can be answered efficiently. We present a new polynomial time algorithm in `RPNI`-style that learns monadic queries defined by deterministic NSTTs from completely annotated examples, where all selected nodes are distinguished.

In practice, users prefer to provide partial annotations. We propose to account for partial annotations by intelligent tree pruning heuristics. We introduce *pruning* NSTT*s* —a formalism that shares many advantages of NSTTs. This leads us to an interactive learning algorithm for monadic queries defined by pruning NSTTs, which satisfies a new formal active learning model in the style of Angluin (1987).

We have implemented our interactive learning algorithm integrated it into a visually interactive Web information extraction system—called `SQUIRREL`—by plugging it into the Mozilla Web browser. Experiments on realistic Web documents confirm excellent quality with very few user interactions during wrapper induction.

**Keywords** Web information extraction · Wrapper induction · Grammatical inference · Tree automata · Monadic queries

**Editor:** Georgios Paliouras and Yasubumi Sakakibara

J. Carme · R. Gilleron · A. Lemay (✉) · J. Niehren
INRIA Futurs and Lille University, LIFL, Mostrare project
e-mail: aurelien.lemay@univ-lille3.fr

## 1. Node selection tasks in Web information extraction

The Web's information overload calls for intelligent services that provide high-quality collected and value-added information. Information from diverse sources sites relevant to the service needs to be integrated and made accessible in a uniform manner. Web documents in HTML are ill-suited for automated processing, which nevertheless is an inevitable prerequisite. This problem is addressed by much work on so-called *Web wrappers*. These are programs that extract the relevant information from HTML documents and translate it into a more machine-friendly format.

Manual wrapper programming is often not manageable. Even for experts, this task remains time consuming and error prone. Furthermore, wrappers must frequently be adapted to changes of the document sources. This is why new tools have been proposed that assist humans in wrapper creation. These are of two kinds:

- *Machine learning techniques* are used for inducing *string based wrappers*. Most systems (Kushmerick, 1997; Hsu & Dung, 1998; Freitag & McCallum, 1999; Freitag & Kushmerick, 2000; Kushmerick, 2000; Muslea, Minton, & Knoblock, 2001; Chidlovskii, 2001) rely on finite-state approaches (patterns, automata and transducers, Hidden Markov Models). See in Kushmerick (2002) for a comparison.
- *Visual specification* of *node selection queries in trees* (Baumgartner, Flesca, & Gottlob, 2001; Gottlob & Koch, 2002). Suitable query specification languages behind the graphical interface are monadic Datalog or similar tree logics.

Some of the most recent approaches propose machine learning techniques for inducing *tree-based wrappers*. One of them is multi-view inductive logic programming (Cohen, Hurst, & Jensen, 2003) which enriches string-based machine learning techniques by tree views. Another is based on finite-state techniques (Kosala et al., 2003; Raeymaekers, Bruynooghe, & Van den Bussche, 2005). This approach differs from ours in that it represents monadic queries by a local tree languages. Induction starts from learning algorithms from positive examples. For a more detailed comparison see Section 6.

In the present article—extending on Carme, Lemay, and Niehren (2004a) and Carme et al. (2005)—we combine features of both, machine learning techniques and visual specification. We define an interactive setting for inducing tree based wrappers. We are interested in machine learning techniques that can infer node selection queries in trees. We consider an HTML document as a tree structure obtained by parsing the document. An example for a rendered HTML document is given in Fig. 1, and its parse tree in Fig. 2. The HTML document contains the email addresses of three persons, called Ingo Bruss, Trevor Bruss, and Trevor Bruss again, each of which is described in its own HTML table. The names of these persons as well as their email addresses are contained in hyperlinks, which correspond to nodes of the parse tree that are labeled by tag `A`.

Suppose that we want to extract all email addresses in this HTML document. With respect to the parse tree, this wish can be expressed by the monadic node selection query that selects all hyperlinks in the last row of a table. In XPath like notation (but with additional axes), this query can be defined as follows:

$$\texttt{/descendant-or-self::TABLE/lastchild::*/descendant::A/child::*}$$

Start at the root, go downwards to a node labeled by TABLE, move to its last child, then move to some A-labeled offspring and finally select one of its children. The same query can

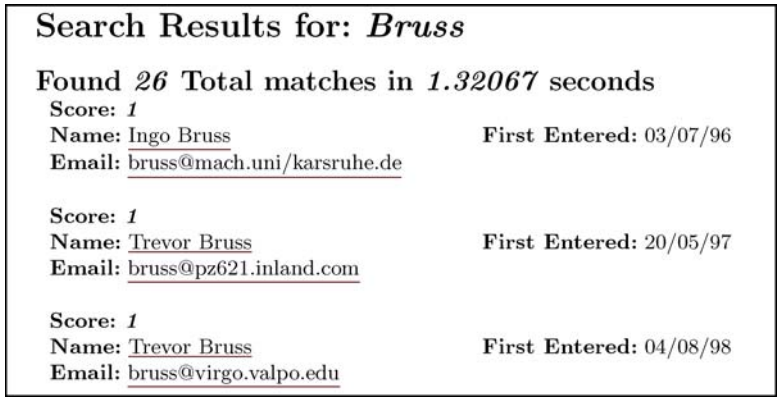
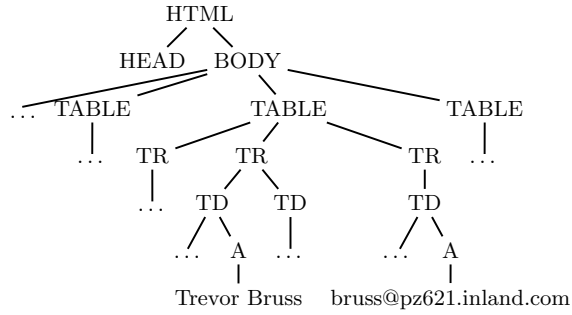



**Fig. 1** A rendered HTML document returned by some Web search machine

**Fig. 2** The parse tree of the HTML document in Fig. 1




be expressed by a conjunctive formula of first-order logic with one free node variable $x$:

$$\{x \mid \exists x_1 \exists x_2 \exists x_3.\, \mathtt{label}_{\mathtt{TABLE}}(x_1) \wedge \mathtt{lastchild}(x_1, x_2) \wedge \mathtt{descendant}(x_2, x_3) \wedge \mathtt{label}_{\mathtt{A}}(x_3) \wedge \mathtt{child}(x_3, x)\}$$

One might opt for a more precise query that further constrains the path from the TABLE to the A labeled node. For tree structures generated by the same search machine, however, such a refinements seem unnecessary to correctly select the informative nodes.

Another task of interest might be to extract all pairs of names and email addresses. This binary query can be expressed by a conjunctive first-order formula with two free variables. In what follows, however, we will restrict ourselves to monadic queries. This is because most existing systems rely on monadic queries for treating more complex binary or $n$-ary cases (Baumgartner, Flesca, & Gottlob, 2001; Muslea, Minton, & Knoblock, 2001; Kosala et al., 2003).

We thus consider the problem of extracting sets of informative nodes in trees, that are generated homogeneously by some back-end database or search machine. We fix a formal query representation language and split the selection task into two subproblems:

find a representation of an appropriate monadic query for informative nodes in trees; here we want to use machine learning techniques, rather than manual query specification or programming.

2. answer the query in a tree: given a tree and a query definition, compute the set of nodes that the query selects in the tree.

Suitable query representation languages thus need to satisfy a number of constraints.

*Expressiveness*: All useful monadic queries should be definable in the chosen query representation language. Monadic second-order logic (MSO) is the yardstick for measuring the expressiveness of query languages for trees and thus wrapper language in information extraction from tree structured documents (Gottlob & Koch, 2002). It allows to define all regular node selection queries in trees, which is adequate for many information extraction tasks.

*Efficiency*: We need efficient algorithms for answering monadic queries given a tree a representation of the query in our language.

*Learnability*: The objective is to induce monadic queries from trees in some of the nodes are annotated that are to be extracted. In the simpler case, we consider completely annotated Web pages in which all nodes are annotated that are subject to extraction. Because complete annotation may be unfeasible, we also consider partial annotations. Therefore we need to consider the problem of learning monadic queries from completely or partially annotated trees.

In this article, we propose two new query representation formalisms for monadic queries that are based on tree automata, *Node Selecting Tree Transducers (*NSTT*s)* and *Pruning Node Selecting Tree Transducers (*PNSTT*s)*. We introduce NSTTs and PNSTTs in Section 2, with the objective to illustrate the basic ideas and concepts related to tree automata for unranked trees and monadic queries by examples.

In Section 3, we show that these query representation languages satisfy some of the algorithmic learnability requirements. What we need is an efficient test to determine whether a tree automaton is functional and thus an NSTT, or cut-functional and thus a PNSTT. The algorithm for cut-functionality is elaborated in the appendix (Section A).

Our results on efficiency and expressiveness of NSTTs and PNSTTs are postponed to the appendix as well (Sections B and C). There, we show that monadic queries defined by deterministic NSTTs or PNSTTs can be answered efficiently with linear time combined complexity. We then show that this holds even though deterministic NSTTs and PNSTTs capture the class of monadic queries definable in MSO. Both query languages NSTTs and PNSTTs are thus unrestricted from the view point of expressiveness.[1]

The subsequent sections are devoted to learning algorithms for node selection queries, that are represented by NSTTs or PNSTTs. In Section 4, we present a new polynomial time algorithm in RPNI-style that we call tRPNI which learns monadic queries defined by deterministic NSTTs from completely annotated examples (for the target query). It relies on a subprocedure for testing membership to the class of NSTTs, i.e. for testing functionality.

In Section 5, we model the interactive process of learning monadic queries as active learning in the style of Angluin (1987), in order to measure the user's effort during the interaction. Only partially annotated trees are available in interactive Web information extraction. We solve this problem by using intelligent tree pruning heuristics and we present a practically feasible interactive learning algorithm for monadic queries defined by PNSTTs. It relies on a sub-procedure for testing membership to the class of PNSTTs, i.e. cut-functionality.

[1] Completeness with respect to expressiveness and learning are independent questions. Our learning algorithm tRPNI for NSTTs is a complete, while our algorithm $\text{RPNI}_{\text{prune}}$ inducing PNSTTs is not (for realistic pruning functions).

Finally, Section 6 discusses the integration of our learning algorithms into visually interactive Web information extraction. We have implemented our core learning algorithm in OCaml and have turned it into a visually interactive Web information extraction system—called SQUIRREL—by integration into the Mozilla Web browser. Experiments on realistic Web documents confirm excellent quality with very few user interactions—annotations and corrections—during wrapper induction.

## 2. Node selecting tree transducers: NSTTs and pNSTTs

Parse trees of HTML or XML documents are sibling ordered unranked trees whose nodes may have an unbounded list of children. In this section, we develop query representation languages for monadic node selection queries in unranked trees, that we will use for query induction by methods of grammatical inference. We rely on tree automata for unranked trees. We propose two kinds of tree automata, that we call *node selecting tree transducers (*NSTT*s)* and *pruning node selecting tree transducers (*PNSTT*s)*.

We start with a gentle introduction to NSTTs and PNSTTs, whose objective is to illustrate the basic ideas and concepts related to tree automata for unranked trees and monadic queries by examples. Algorithmic learnability aspects will be discussed in Section 3. Questions on efficiency and expressiveness for NSTTs and PNSTTs are answered in the appendix (Sections B and C).

### 2.1. Monadic queries in unranked trees

Let $\mathbb{N} = \{1, 2, \ldots\}$ be the set of natural numbers without 0 and $\mathsf{Bool} = \{0, 1\}$ be the set of Booleans. Let $\Sigma$ be a finite set, the set of *node labels*, also called the *signature*.

An *unranked tree* over $\Sigma$ is a finite tree with $\Sigma$-labeled nodes whose edges are totally ordered from the left to the right. An unranked tree $t$ over $\Sigma$ is a pair consisting of a label $a \in \Sigma$ and a possibly empty list of unranked trees $t_1, \ldots, t_n$: the following abstract syntax:

$$t ::= a(t_1, \ldots, t_n) \qquad \text{where } n \in \mathbb{N} \cup \{0\} \text{ and } a \in \Sigma$$

As a running example, we consider the following unranked tree with labels in $\Sigma = \{\mathtt{L}, \mathtt{F}, \mathtt{A}, \mathtt{H}, \mathtt{W}\}$:

$$\mathtt{films} = \mathtt{L}(\mathtt{F}(\mathtt{A}, \mathtt{H}), \mathtt{F}(\mathtt{A}, \mathtt{W}), \mathtt{F}(\mathtt{A}, \mathtt{H}))$$

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 (A) represents the actor lists of these films (which would rather expand to a larger subtree in realistic examples). We draw trees as graphs, as for instance in Fig. 3.

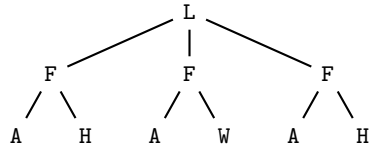



**Fig. 3** The unranked tree `films = L(F(A, H), F(A, W), F(A, H))`. We want to query for the actor lists (A) in the films by Hitchcock (H)

As in this example, we will systematically ignore textual contents of HTML or XML documents in this article. This design decision clearly simplifies our tasks, which nevertheless remains useful for practical applications to Web information extraction, as we will see.

We write $\texttt{nodes}(t)$ for the set of nodes of a tree $t$. We identify a node with its relative address, so that $\texttt{nodes}(t) \subseteq \mathbb{N}^*$. The node 21, for instance, is the first child of the second child of the root. In our example tree `films`, this node is labeled by `A`. We write $t(v)$ for the the label of some $v \in \texttt{nodes}(t)$, for instance:

$$\texttt{films}(21) = \texttt{A}$$

We say that two trees have *the same shape* if they have the same set of nodes. This definition works, since we identify nodes with their relative address from the root.

*Definition 1.* A *monadic query* in unranked $\Sigma$-trees is a function $q$ that maps unranked trees $t$ over $\Sigma$ to subsets of their nodes:

$$q(t) \subseteq \texttt{nodes}(t)$$

The function `nodes` itself, for instance, is a monadic query that maps a tree $t$ to the set of all its nodes. A less trivial example is the query `actors_hitchcock` which asks for the set of all actor lists in films by Hitchcock. Applied to the example tree `films`, this query yields the following set of nodes:

$$\texttt{actors\_hitchcock}(\texttt{films}) = \{11, 31\}$$

Each actor list of a Hitchcock film corresponds to some node labeled by $A$, that has a parent labeled by $F$, and a next sibling to the right that is labeled by $H$. Furthermore, the $F$-parent should belong to a list of films, so that its own parent is labeled by $L$ and all its siblings by $F$.

## 2.2. Tree automata for unranked trees

Tree automata are machines that recognize sets of trees. Tree automata are traditionally developed for binary or ranked trees (Comon et al., 1997), and indeed, the situation gets different for unranked trees.

A large number of different automata notions for unranked trees and querying have been proposed recently. Besides so called *unranked tree automata (UTAs)* by Brüggemann-Klein, Wood, and Murata (2001) that are most popular in the database community (Libkin, 2005) and identical with the much earlier *pseudo-automata* by Thatcher (1967), there are so called *query automata* (Neven & Schwentick, 2002), forest grammars of Neumann and Seidl (1998), automata on binary encodings of unranked trees (Frick, Grohe, & Koch, 2003) and the algebraic recognizably notion of *stepwise tree automata* by Carme, Niehren, and Tommasi (2004b).

With respect to query induction as proposed in this article, we need an automata notion for unranked trees that comes with a good notion of bottom-up determinism, so that the Myhill-Nerode theorem on minimization holds. Martens and Niehren (2006) discuss the available alternatives. They show that unique minimization fails for deterministic UTAs, and argue that stepwise tree automata yield smaller minimal automata than the inverted standard binary encoding (up to a linear factor depending on the size of the alphabet) and *parallel UTAs*

(Raeymaekers & Bruynooghe, 2004; Cristau, Löding, & Thomas, 2005). We thus choose to work with stepwise tree automata.

Stepwise tree automata can be understood as standard tree automata that operate on binary encodings of unranked trees (not the standard encoding however). This fact will come in handy for our more formal results on NSTTs and PNSTTs in Section 3. In this section, however, we will use finite word automata syntax to present stepwise tree automata. Semantically, however, they behave such as tree automata. The relationship to standard tree automata will be clarified on need (in Section 3.2).

*Definition 2.* A *stepwise tree automaton* for unranked trees over $\Sigma$ is a finite word automaton $A$ with $\texttt{alphabet}(A) = \texttt{states}(A) \uplus \Sigma$ such that

- there is a unique initial state: $\texttt{initial}(A) = \{q_I\}$.
- if $q_0 \xrightarrow{a} q'$ in $\texttt{rules}(A)$ for some $a \in \Sigma$ then $q_0 = q_I$, and
- no transition $q_1 \xrightarrow{q_2} q$ in $\texttt{rules}(A)$ satisfies $q = q_I$ nor $q_2 = q_I$.

A run $r$ of a stepwise tree automaton $A$ on an unranked tree $t$ labels the nodes of $t$ by states of $A$. It is a function of type $r : \texttt{nodes}(t) \to \texttt{states}(A)$ that can be understood as a tree over $\texttt{states}(A)$ having the same shape as $t$. Value $r(v)$ is the state that run $r$ assigns to node $v \in \texttt{nodes}(t)$. If $v$ has $n$ children $v1, \ldots, vn$ then $r(v)$ is a state obtained by running $A$ on the following word:

$$t(v)\, r(v1)\, \ldots\, r(vn)$$

This word starts with the label of $v$ in $t$ and continues by the sequence of states that $r$ assigns to the children of $v$. A run is *successful* if it maps the root of $t$ to a final state of $A$. A stepwise tree automaton $A$ accepts an unranked tree $t$ if there exists a successful run of $A$ on $t$. The language $L(A)$ of a stepwise tree automaton $A$ is the set of all unranked trees that $A$ accepts. An example for a stepwise tree automaton is given in Fig. 4. This automaton accepts all lists containing films by Hitchcock and Wenders. The `films` tree annotated by a run of this automaton is drawn in Fig. 4.

*Definition 3.* A stepwise tree automaton $A$ is (*bottom-up*) *deterministic*, if it is a left-to-right deterministic finite automaton, i.e., for ever every $q \in \texttt{states}(A)$ and $a \in \texttt{alphabet}(A)$ there exists at most one $q'$ in $\texttt{states}(A)$ such that $q \xrightarrow{a} q'$ belongs to $\texttt{rules}(A)$.

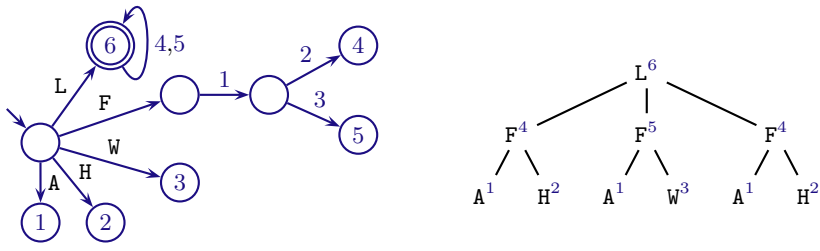



**Fig. 4** A stepwise tree automaton that accepts film lists, and the `films` tree annotated by a successful run of this automaton. State 6 of the root is obtained by evaluating the word L454 by the automaton

What might be less clear at the time being is that every stepwise tree automaton can be determinized without altering its tree language, and that all regular languages of unranked trees are recognized by a unique minimal deterministic stepwise tree automaton (up to isomorphism). This will become obvious from the tree automata perspective adopted in Section 3.

### 2.3. NSTTs for unranked trees

The next goal is to use stepwise tree automata in order to define monadic queries in unranked trees. There are many possibilities to do so, NSTTs provide one of them, pNSTTs another. Two further methods will be discussed and used in Section B.

An *annotated tree* over $\Sigma$ is a tree over $\Sigma \times \mathsf{Bool}$. Every annotated tree can be decomposed into a product $t \times \beta$ between a $\Sigma$-tree $t$ and a $\mathsf{Bool}$-tree $\beta$ of the same shape. For convenience, we will often write $a_b$ for pairs $(a, b)$. For instance:

$$\mathtt{L(F(A,H))} \times \mathtt{0(0(1,0))} \;=\; \mathtt{L_0(F_0(A_1,H_0))}$$

A *completely annotated tree for a query* $q$ in $\Sigma$-trees is an annotated tree of the form:

$$t \times q(t)$$

for some $\Sigma$-tree $t$. Here, we freely consider the set $q(t)$ as a $\mathsf{Bool}$-tree of the shape of $t$, which relabels selected nodes of $t$ to 1 and all others to 0. Examples for three completely annotated trees for the query `actors_hitchcock` are given in Fig. 5.

The idea of NSTT*s* is to represent a monadic query $q$ by a tree automaton that recognizes the set of completely annotated trees for $q$ except for some $t \times q(t)$ where $q(t) = \emptyset$. Such languages $L$ of annotated trees are *functional* in that for every $\Sigma$-tree $t$ there exists at most one Boolean tree $\beta$ such that $t \times \beta \in L(A)$ (namely $\beta = q(t)$).

*Definition 4.* An NSTT over $\Sigma$ is a tree automaton $A$ over $\Sigma \times \mathsf{Bool}$ that recognizes a functional language of annotated trees. Every NSTT $A$ defines a monadic query $q_A$ such that for all unranked $\Sigma$-trees $t$:

$$q_A(t) = \{v \mid \exists \mathsf{Bool}\text{-tree } \beta \text{ s.t. } t \times \beta \in L(A), \beta(v) = 1\}$$

As example, we present in Fig. 6 a deterministic NSTT that defines the query `actors_hitchcock`. Let us recall that we freely write $a_b$ instead of $(a, b)$.

A query $q$ is *represented by an* NSTT $A$ if $A$ defines the query $q$, i.e. $q_A = q$. As we do not care about trees $t$ such that $q(t) = \emptyset$, a query $q$ may be represented by several NSTTs. Two NSTTs $A$ and $A'$ are said to be *equivalent* if $q_A = q_{A'}$.

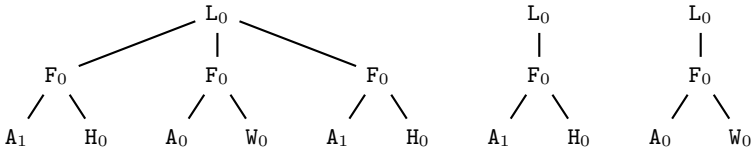



**Fig. 5** Three film lists completely annotated by the query `actors_hitchcock`

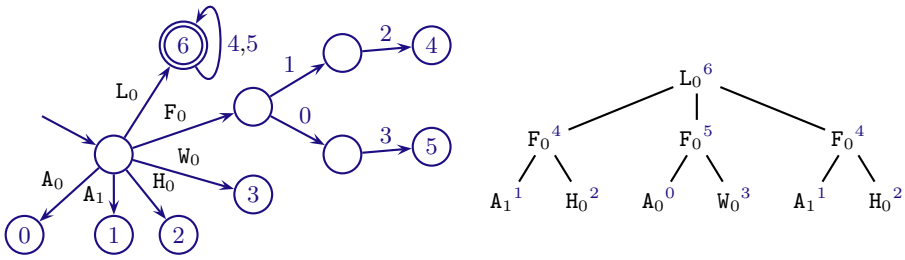



**Fig. 6** A deterministic NSTT defining the query `actors_hitchcock` and one of its successful runs

## 2.4. Pruning NSTTs for unranked trees

One of the disadvantages of NSTTs is that they need to model all subtrees, even those that are irrelevant for node selection. The NSTT in Fig. 4, for instance, has rules to recognize films that are not directed by Hitchcock (but by Wenders). In Web information extraction, such information is typically not available or difficult to learn from the very few given examples.

We propose PNSTTs to solve this problem. These are machines that operate like NSTTs except they first prune trees non-deterministically and then select nodes from the pruned tree. This way, they are able to ignore all pruned subtrees.

Let T be a special constant, that we use to mark leaves in pruned trees where complete subtrees have been cut off. A *pruned tree* over a signature $\Sigma$ is an unranked tree over the signature

$$\Sigma_{\mathrm{T}} = \Sigma \cup \mathrm{T}$$

whose inner nodes are labeled in $\Sigma$. Only leaves of pruned trees may be labeled by T. For every $\Sigma$-tree $t$, let $\mathtt{cuts}(t)$ be the set of $\Sigma_T$-trees obtained by pruning some subtrees of $t$, i.e., by substituting some complete subtrees of $t$ by $T$. Two pruned trees $t_1$ and $t_2$ are said to be *compatible* if they can be obtained by pruning from some common third tree, i.e., if there is a tree $t$ such that $t_1, t_2 \in \mathtt{cuts}(t)$.

See Fig. 7 for some examples of pruned annotated trees, which have the signature $(\Sigma \times \mathsf{Bool})_T$. It will be convenient to identify pruned annotated trees over $(\Sigma \times \mathsf{Bool})_T$ with trees over $(\Sigma \times \mathsf{Bool}) \uplus \{T_T\}$, i.e., we implicitly assume $T = T_T$ when decomposing a tree over $(\Sigma \times \mathsf{Bool})_T$ into a product $t \times \beta$ of trees $t$ over $\Sigma_T$ and a tree $\beta$ over $\mathsf{Bool}_T$. Intuitively, the annotation of $T$ by $T$ means that we don't know whether nodes in pruned subtrees will be selected or not.

We call a language $L$ of $(\Sigma \times \mathsf{Bool})_T$-trees *cut-functional* if for all $t_1 \times \beta_1, t_2 \times \beta_2 \in L$ such that $t_1$ and $t_2$ are compatible, it holds that $\beta_1$ and $\beta_2$ are compatible. This means that different prunings $t_1$ and $t_2$ of the same third tree $t$ will never carry contradictory information. A $\Sigma_T$-tree $t$ is called *unpruned* if $T$ does not occur in the set of node labels used by $t$. Note that cut-functional languages of unpruned trees (and thus with signature $\Sigma \uplus \mathsf{Bool}$) are functional.

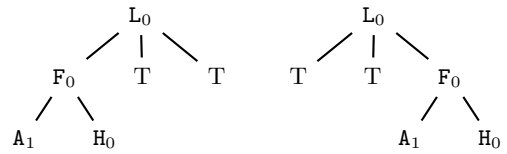



**Fig. 7** Two trees obtained by pruning the tree `films` annotated by query `actors_hitchcock`. The Boolean annotations are compatible with this query

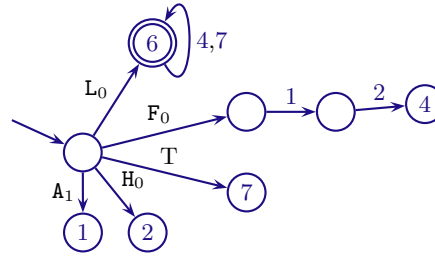



**Fig. 8** A deterministic PNSTT defining the query `actors_hitchcock`

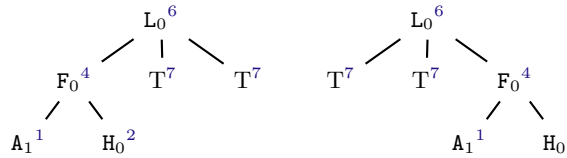



**Fig. 9** Two runs of different prunings of `films` by the PNSTT in Fig. 8. Each of them selects one of the A-nodes in Hitchcock's films

Consider for instance the two annotated prunings of `films` in Fig. 7. Their Boolean annotations are compatible in that both of them require the root to be unselected. For all other nodes, there exist no overlapping annotations. The fact that the left-most (resp. right-most) `A`-node is to be selected is available on the left-most (resp. right-most) pruning only.

*Definition 5.* A PNSTT over $\Sigma$ is a tree automaton over $(\Sigma \times \mathsf{Bool})_T$ whose language is cut-functional. Every PNSTT $A$ defines the query $q_A$ such that for all $\Sigma$-trees $t$:

$$q_A(t) = \{v \in \mathtt{nodes}(t) \mid \exists\, t' \in \mathtt{cuts}(t), \exists \beta' :\ t' \times \beta' \in L(A), \beta'(v) = 1\}$$

The nodes of $q_A(t)$ can thus be computed non-deterministically, by guessing a pruning of $t'$ of $t$ and inspecting the Boolean annotation $\beta'$ such that $t' \times \beta' \in L(A)$, which is unique by cut-functionality if it exists.

A deterministic PNSTT that recognizes the actors lists of Hitchcock's films is given in Fig. 8. Indeed, it leaves the structure of all films by other directors than Hitchcock undefined, since these can be pruned to $T$. Two run of tree Fig. 8 are given Fig. 9.

A query $q$ is *represented by a* PNSTT $A$ if $A$ defines the query $q$, i.e. $q_A = q$. A query $q$ may be represented by several PNSTTs.

## 3. Testing functionality in polynomial time

Queries defined by NSTTs and PNSTTs have three important algorithmic or descriptive properties that render them suitable for interactive query induction.

*Membership*: whether a deterministic tree automaton over $\Sigma \times \mathsf{Bool}$ is functional, and hence an NSTT can be decided in polynomial time. Whether a tree automaton over $(\Sigma \times \mathsf{Bool})_T$ is cut-functional and thus a PNSTT can be decided in polynomial time.

*Efficiency*: monadic queries defined by NSTTs or PNSTTs can be answered in polynomial time

*Expressiveness*: deterministic NSTTs can express all monadic second order definable monadic queries, as well as deterministic PNSTTs.

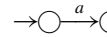
In this section, we prove the first property, by presenting efficient algorithms for membership testing, on which our learning algorithms will rely. Efficiency will be treated in Section B. It is fundamental because correctness of the hypothesis NSTT or PNSTT must be checked frequently in the interactive setting. Also, an information extraction program must be efficient. Expressiveness will be treated in Section C. We show that deterministic NSTTs and PNSTTs can express all MSO-definable queries. The learning algorithm for NSTTs will be able to infer all MSO-definable queries, when given enough examples (which is unrealistic in practice). The choice of a practical pruning function will restrict the class of learnable queries by the learning algorithm of PNSTTs.

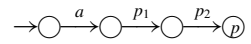
### 3.1. Tree automata for binary trees

Properties depending on determinism and functionality are easier to establish for standard tree automata on binary trees. As we will show, they can be lifted to stepwise tree automata by exploiting a particular binary encoding of unranked trees, called Currying in Carme, Niehren, and Tommasi (2004b). Note that the notion of determinism for tree automata on unranked trees works out smoothly for stepwise tree automata in contrast to more standard notions of tree automata for unranked trees (Martens & Niehren, 2006).

Standard tree automata (Comon et al., 1997) are traditionally defined for binary trees, i.e., unranked trees where every node has either no or exactly 2 children. For sake of simplicity, we will develop our algorithms for binary trees, and then lift them to unranked trees via binary encoding.

A *tree automaton* $A$ for binary trees over some signature $\Sigma$ consists of a finite set $\texttt{states}(A)$, a subset $\texttt{final}(A) \subseteq \texttt{states}(A)$, and a set of rules of two kinds where $a \in \Sigma$ and $p, p_1, p_2 \in \texttt{states}(A)$.

$a \to p$ instead of 

$a(p_1, p_2) \to p$ instead of 

Standard tree automata can be seen as particular stepwise tree automata, except that they decompose binary rules into three steps.

As a consequence, the notions of runs, successful runs, and accepted tree languages carry over immediately. We write $\texttt{runs}_A(t)$ for the set of runs of automaton $A$ on tree $t$ and $\texttt{succ\_runs}_A(t)$ for the subset of successful runs. The notion of bottom-up determinism carries over from stepwise to standard tree automata, too. A tree automaton is bottom-up deterministic if none of its rules have the same left hand side. It is well known that every standard tree automaton can be determinized without affecting its tree language. Further notions that carry over to binary trees without any changes are functionality, cut-functionality, NSTTs, and PNSTTs.

### 3.2. Binary encoding of unranked trees

We next relate deterministic stepwise tree automata for unranked trees to deterministic standard tree automata over binary trees. This shows that subsequent learnability results for deterministic standard tree automata can be lifted to deterministic stepwise tree automata. The base idea is to encode unranked trees into binary trees by using Currying. The better known first-child next-sibling encoding fails to do this job; it does not preserve determinism.

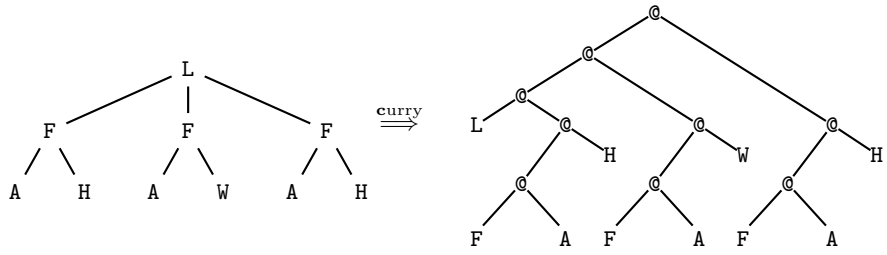



**Fig. 10** Curried binary encoding: curry(L(F(A, H), F(A, W), F(A, H))) = L@(F@A@H)@(F@A@W)@(F@A@H)

Some more care is needed when determinizing stepwise tree automata. The usual determinization algorithm for finite automata fails since it alters the tree language of a stepwise tree automaton. Instead, a stepwise tree automaton needs to be understood as a standard tree automata operating on Curryied encodings. Determinisation for standard tree automata is then doing the job.

For applying Currying we consider unranked trees as lambda terms. The term L(F, F, F) for instance, is a lambda term that describes the application of function L to a triple of arguments. Its Curried encoding is the binary tree (((L@F)@F)@F) which describes the application of function L to its arguments stepwise one by one. The Curried binary encoding of the unranked tree films is displayed in Fig. 10.

Stepwise tree automata $A$ over $\Sigma$ can be understood as syntax for standard tree automata $A^b$ for binary trees over $\Sigma \uplus \{@\}$, which operates on Curried binary encodings of unranked trees. The translation is as follows, where $a \in \Sigma$ and $p_0 \in \texttt{initial}(A)$, and $p_1, p_2, p \in \texttt{states}(A) - \texttt{initial}(A)$:

$$p_0 \xrightarrow{a} p \in \texttt{rules}(A) \text{ iff } a \to p \in \texttt{rules}(A^b)$$
$$p_1 \xrightarrow{p_2} p \in \texttt{rules}(A) \text{ iff } @(p_1, p_2) \to p \in \texttt{rules}(A^b)$$

The translation of the stepwise automaton in Fig. 4 is given in Fig. 11. Generally, it holds that:

$$A \text{ is deterministic} \;\Leftrightarrow\; A^b \text{ is deterministic.}$$

Labels: $\{@, \texttt{A}, \texttt{H}, \texttt{W}, \texttt{F}, \texttt{L}\}$
States: $\{1, 2, 3, 4, 5, 6, 8, 9\}$
Rules:

| | |
|---|---|
| $\texttt{A} \to 1$ | $@(8,1) \to 9$ |
| $\texttt{H} \to 2$ | $@(9,2) \to 4$ |
| $\texttt{F} \to 8$ | $@(6,4) \to 6$ |
| $\texttt{L} \to 6$ | $@(6,5) \to 5$ |
| $\texttt{W} \to 3$ | $@(9,3) \to 5$ |
| | $@(6,5) \to 6$ |

Final: $\{6\}$

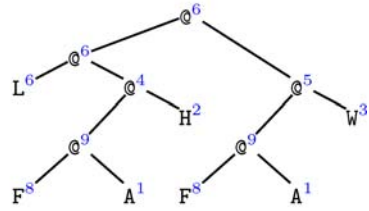



**Fig. 11** A standard tree automaton for binary encodings of film lists, corresponding to the stepwise tree automaton in Fig. 4

This follows immediately from the definitions. As an immediate consequence, it follows that every stepwise tree automaton can be determinized. Furthermore, the language of both automata correspond modulo Currying, i.e., for all unranked trees $t$ over $\Sigma \uplus \{@\}$:

$$t \in L(A) \;\Leftrightarrow\; \mathtt{curry}(t) \in L(A^b)$$

This is because both automata produce runs that correspond precisely except for intermediate results stored in @-labeled nodes (which could be annotated to the edges of unranked trees). Since $\mathtt{states}(A^b) = \mathtt{states}(A) - \mathtt{initial}(A)$, it also follows that there exists unique minimal deterministic stepwise tree automata for all regular languages of unranked trees.

The correspondence lifts to monadic queries defined by NSTTs or pNSTTs even though this needs a little care. An NSTT $A$ over $\Sigma$ is a tree automaton over $\Sigma \times \mathsf{Bool}$, so that $A^b$ is an automaton over $(\Sigma \times \mathsf{Bool}) \uplus \{@\}$. Unfortunately, $A^b$ is not an NSTT over $\Sigma \times \{@\}$, but this problem can be resolved by replacing @ by $@_0$, which means that we do not want to extract any auxiliary nodes introduced by the binary encoding, which is of course consistent with our intuition. Modulo the equation $@ = @_0$ it holds for every tree automaton $A$ over $\Sigma \times \mathsf{Bool}$, and $\Sigma$-tree $t$:

$$\begin{aligned} A \text{ is functional} &\;\Leftrightarrow\; A^b \text{ is functional} \\ q_A(t) &\;=\; q_{A^b}(\mathtt{curry}(t)) \end{aligned}$$

Hence, $A$ is an NSTT for unranked trees over $\Sigma$ if and only if $A^b$ is an NSTT for binary trees over $\Sigma \uplus \{@\}$, and that both automata define the same query modulo the natural correspondence between nodes in unranked trees to leafs in binary encodings. As a consequence, we can reduce functionality testing and NSTT query answering on unranked trees to the respective problems on binary trees.

The same arguments carry over to pNSTTs without any further changes. Modulo the equation $@ = @^0$ it holds for every tree automaton $A$ over $(\Sigma \times \mathsf{Bool})_T$, and $\Sigma$-tree $t$:

$$\begin{aligned} A \text{ is cut-functional} &\;\Leftrightarrow\; A^b \text{ is cut-functional} \\ q_A(t) &\;=\; q_{A^b}(\mathtt{curry}(t)) \end{aligned}$$

### 3.3. Testing functionality and cut-functionality

We next show how to test whether a tree automaton for binary trees over $\Sigma \times \mathsf{Bool}$ is functional. As argued above, this is sufficient to solve the same problem for unranked trees.

We reduce functionality testing to unambiguity testing, a problem for tree automata that can be solved in polynomial time (Seidl, 1989).

A tree automaton is *unambiguous* if it permits at most one successful run per tree. Note that every deterministic tree automaton is unambiguous, since it permits at most one run per tree, may it be successful or not.

The *projected automaton* $\pi(A)$ over $\Sigma$ has $\mathtt{states}(\pi(A)) = \mathtt{states}(A) \times \mathsf{Bool}$, $\mathtt{final}(\pi(A)) = \mathtt{final}(A) \times \mathsf{Bool}$, and the rules are given by the following inference scheme where $a \in \Sigma$ and $b, b_1, b_2 \in \mathtt{Bool}$:

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

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

**Lemma 2.** *A deterministic tree automaton A over* $\Sigma \times \mathsf{Bool}$ *is functional and thus an* NSTT *if and only if its projection* $\pi(A)$ *to* $\Sigma$ *is unambiguous.*

**Proof:** For the one direction, let $A$ be a deterministic NSTT. 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 1.** *Whether a deterministic tree automaton over* $\Sigma \times \mathsf{Bool}$ *is functional and hence an* NSTT *can be tested in cubic time.*

It is sufficient to prove this proposition for standard tree automata over binary trees, so that it carries over to stepwise tree automata for unranked trees.

**Proof:** Let $A$ be a deterministic tree automaton over $\Sigma \times \mathsf{Bool}$. By Lemma 2 it is sufficient to compute the $\Sigma$-projection $\pi(A)$ and to test whether this automaton is unambiguous. This can be done in cubic time. □

It remains to show that we can equally decide membership to the class of deterministic PNSTT in polynomial time. This is stated by the following proposition that is proved in Section A.

**Proposition 2.** *Cut-functionality of deterministic* PNSTT*s can be tested in cubic time.*

## 4. Learning NSTTs from completely annotated examples

We have shown that monadic node selection queries can be represented by deterministic NSTTs which are tree automata. It is known that regular tree languages represented by deterministic bottom-up tree automata are learnable from polynomial time and data (Oncina & García, 1993). But this result holds only when positive and negative examples are available, i.e. trees belonging to the target language or not. In our query learning framework, however, completely annotated trees for the target query will be available instead. These express a number of positive and implicit negative examples. We will prove in this section that monadic node selection queries represented by NSTTs are learnable from polynomial time and data, provided that the data contains completely annotated examples for the target query. The difficulty is to show, that completely annotated examples are indeed equivalent to arbitrary samples of positive and negative examples.

### 4.1. Identification from polynomial time and data

We recall the learning model from polynomial time and data (Gold, 1967, 1978; de la Higuera, 1997), and adapt it to the problem of tree language inference using deterministic tree automata. An example is a couple $(t, b)$ where $t$ is a tree and $b$ is its Boolean label. Positive examples are labeled by 1 and negative examples by 0. A (tree) sample is a finite set of examples. A tree automaton $A$ is compatible with an example $(t, b)$ if $(t \in L(A)) \Leftrightarrow b$, it is compatible with

a sample if it is compatible with all its examples. We also say in this case that the sample is consistent with $A$. The *size* of a sample is the total number of nodes of all trees in the sample.

*Definition 6.* Tree languages represented by a class of tree automata $\mathcal{C}$ are said to be *identifiable from polynomial time and data* if there exist two polynomials $p_1$ and $p_2$ and an algorithm learner such that:

- with as input a sample $S$ of size $m$, learner returns a tree automaton $A \in \mathcal{C}$ compatible with $S$ in $O(p_1(m))$ time;
- for each tree automaton $A \in \mathcal{C}$ of size $n$, there exists a sample—called the characteristic sample for $A$—char($A$) of *cardinality* less than $p_2(n)$ such that, with input a sample $S$ that contains char($A$), learner returns a tree automaton $A' \in \mathcal{C}$ equivalent with $A$.

Regular tree languages represented by deterministic tree automata are identifiable from polynomial time and data (Oncina & García, 1993). In the sequel of the paper we mainly use the existence of a polynomial time learning algorithm learner. It is set to be the RPNI algorithm for deterministic tree automata which is defined in Oncina and García (1993) as an extension of the RPNI algorithm for deterministic string automata (Oncina & Garcia, 1992).

The reader should note that the model for tree languages differs from the model for string languages on one important point: it is required that the *cardinality* of the characteristic sample is polynomial in the size of the target automaton whereas for string languages it is required that the *size* of the characteristic sample is polynomial in the size of the target automaton. This is of course weaker and due to the fact that the size of trees—measured as the number of nodes—in the characteristic sample can be exponential. Indeed, consider the tree language containing a single balanced binary tree of height $n$ on the alphabet $\Sigma = \{a, f\}$ where $a$ is a constant and $f$ a binary symbol. The characteristic sample must contain this tree which has $2^{n+1} - 1$ nodes while the corresponding tree automaton has only $n + 1$ states. This blow-up is indeed due to the intrinsic nature of trees. This problem could be solved with a new representation schema for trees in the characteristic sample: represent a tree by a deterministic automaton recognizing the language reduced to this unique tree. The proof is out of scope of the paper because we will only consider the learner defined with the RPNI algorithm presented in Oncina and García (1993).

The learning model for tree languages is not well suited for node selection queries in trees. We can represent a node selection query by a regular tree language over $\Sigma \times \mathtt{Bool}$. But, for learning node selection queries, only completely annotated trees for the target query will be available. Thus, if $q$ is the target query, only positive examples $(t \times q(t), 1)$ will be available. We will refer to them as completely annotated examples. Also, negative examples are useless as they can be deduced from completely annotated ones: for a completely annotated example $t \times \beta$, every other annotated tree $t \times \beta'$ $(\beta' \neq \beta)$ is negative. Two query representations are equivalent if they represent the same query. This leads to the following definition:

*Definition 7.* Node selection queries represented in a class of automata $C$ 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:

- with input a set $S$ of annotated trees of size $m$, learner returns a query representation $A \in R$ in time $O(p_1(m))$ such that $q_A$ is compatible with $S$
- for each automaton $A \in C$ of size $n$, there exists a set of completely annotated examples for $q_A$—called the characteristic sample for $A$—char($A$) of *cardinality* less than $p_2(n)$ such that, for every sample $S$ that contains char($A$), learner returns an automaton $A' \in C$ such that $q_A = q_{A'}$.

We will simply talk of identifiability when we do not impose polynomial constraints.

### 4.2. tRPNI: An RPNI extension for deterministic NSTTs

We now want to prove that node selection queries represented by deterministic NSTTs are identifiable from polynomial time and data. We first show that one can easily deduce identifiability of node selection queries represented by deterministic NSTTs in this framework from the previous result of identifiability of tree languages represented by deterministic automata—polynomial constraints set aside.

Note that the proof presented here leads to an exponential time algorithm—as we will have to enumerate explicitly every negative example corresponding to a different annotation of a completely annotated example. Polynomiality can nevertheless be reached by using the functionality property of query languages in order to avoid this explicit enumeration. This leads us to a new variation of the learning algorithm RPNI which can identify node selection queries represented by NSTTs.

**Lemma 3.** *Node selection queries represented by deterministic* NSTT*s are identifiable from completely annotated examples.*

**Proof:** First note that annotated tree languages represented by deterministic NSTTs are identifiable as they are deterministic tree automata over $\Sigma \times \mathsf{Bool}$. Let learner be an algorithm according to Definition 6 and let char be a function that computes the characteristic sample for any deterministic tree automaton over $\Sigma \times \mathtt{Bool}$ according to Definition 6.

To prove the lemma, we have to prove we can infer an NSTT defining the target query from completely annotated examples. A completely annotated example $t \times \beta$ for a query $q$ defines a positive example $(t \times \beta, 1)$ and negative ones $(t \times \beta', 0)$ for all $\beta' \neq \beta$. Thus, we define a function pn that takes as an input a set $S$ of annotated trees and outputs a sample over $\Sigma \times \mathtt{Bool}$.

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

We now define $\mathsf{learner}'$ and $\mathsf{char}'$ the following way :

$$\mathsf{learner}'(S) = \mathsf{learner}(\mathsf{pn}(S))$$
$$\mathsf{lan}_q = \{t \times q(t) \mid t \text{ is a } \Sigma - \text{tree}\}$$
$$\mathsf{char}'(q) = \{t \times \beta \in \mathsf{lan}_q \mid (t \times \beta', b) \in \mathsf{char}(\mathsf{lan}_q)\}$$

First note that $\mathsf{char}'(q)$ is a set of completely annotated examples for $q$.

Second, note that $\mathsf{char}(\mathsf{lan}_q) \subseteq \mathsf{pn}(\mathsf{char}'(q))$. For positive examples $(t \times \beta', 1) \in \mathsf{char}(\mathsf{lan}_q)$, clearly $t \times \beta' \in \mathsf{char}'(q)$ and $(t \times \beta', 1) \in \mathsf{pn}(\mathsf{char}'(q))$. For negative examples $(t \times \beta', 0) \in \mathsf{char}(\mathsf{lan}_q)$, since $\mathsf{lan}_q$ is defined such that for every $\Sigma$-tree $t$, there is a unique tree $t \times \beta \in \mathsf{lan}_q$ (with $\beta = q(t)$), we have $t \times \beta \in \mathsf{char}'(q)$. Hence $(t \times \beta', 0) \in \mathsf{pn}(\mathsf{char}'(q))$.

Let $q$ be a node selection query and $S \supseteq \mathsf{char}'(q)$ a sample consistent with $\mathsf{lan}_q$. Then $\mathsf{pn}(S) \supseteq \mathsf{char}(q)$ and $\mathsf{pn}(S)$ consistent with $\mathsf{lan}_q$. As learner identifies regular tree languages represented by deterministic tree automata over $\Sigma \times \mathsf{Bool}$, $\mathsf{learner}(\mathsf{pn}(S)) = \mathsf{learner}'(S)$ outputs

a deterministic tree automaton $A$ such that $L(A) = \mathsf{lan}_q$, as required for identifying queries represented by NSTTs. □

The proof is established using the function pn which is clearly not polynomial. To establish identifiability results with respect to the polynomial constraints, we must use a different strategy. *Functionality* is the key notion that allows us to have a polynomial learning algorithm: whenever a learning algorithm for tree languages like RPNI checks consistency on their input samples composed of positive and negative examples, we can here verify that the inferred tree automaton is an NSTT, that is it corresponds to a functional tree language. Indeed, recall that for a functional tree language over $\Sigma \times \mathtt{Bool}$, there is at most one annotated tree implying that all different annotations lead to negative examples.

**Theorem 1.** *Monadic node selection queries represented by deterministic* NSTT*s are identifiable from polynomial time and data from completely annotated examples.*

**Proof:** First, let us recall the RPNI-algorithm (Oncina & García, 1993; Oncina & Garcia, 1992; Lang, 1992). RPNI inputs a sample $S$ of positive and negative examples. It first computes a deterministic automaton which recognizes the set of positive examples in the sample called the initial automaton and denoted NSTT($S$)—it is the largest deterministic tree automaton that recognizes exactly $S$. It then merges states exhaustively in some fixed order. A merging operation applies to the recent deterministic automaton $A$ and two states $q_1, q_2 \in \mathtt{states}(q)$ and returns a deterministic automaton det-merge($A, q_1, q_2$). A deterministic merge is a merge followed by recursive merges needed to 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 det-merge($A, q_1, q_2$) is consistent with all negative examples in the sample. The algorithm tRPNI (described in Fig. 12) behaves as RPNI except that it checks differently whether deterministic merging operation are licensed. It tests whether the language of det-merge($A, q_i, q_j$) is functional. It thereby avoids to enumerate implicit negative examples. And fortunately, we can check for functionality in polynomial time in the automaton size (Proposition 1) and there are at most quadratically many functionality tests. Therefore:

– the polynomial time condition for learner equal to tRPNI is satisfied. Consistency with the input sample is also given: by construction, the inferred automaton always recognizes trees of $S$.
– The characteristic set of completely annotated examples is given in the proof of Lemma 3. Given any sample containing this characteristic set, tRPNI behaves as the algorithm learner' described in this lemma (except that it does not explicitly enumerate negative examples) and therefore induces the target NSTT. The polynomial cardinality of the characteristic set

**Fig. 12** Learning from completely annotated examples

tRPNI $(S)$

$A \leftarrow \mathrm{NSTT}(S)$
**For** $i = 1$ **to** $|\mathtt{states}(A)|$ **do**
  **For** $j = 0$ **to** $i - 1$ **do**
    $A' \leftarrow$ det-merge$(A, q_i, q_j)$
    **If** $A'$ is functional **then** $A \leftarrow A'$
**Output :** $A$

of annotated trees is guaranteed by the polynomial cardinality of the characteristic sample of the target NSTT.

□

## 5. Interactive learning of pNSTTs

The above framework has two main drawbacks:

- completely annotated examples are required, which may be a problem in practical cases where sometimes hundreds of nodes would have to be selected by the user. Therefore, one should deal with examples that are only partially annotated.
- Furthermore, it does not take into account the interactions between the user and the algorithm. The user does not simply provide a set of annotated examples. He should for example be able to give some annotations to the algorithm in order to have a candidate query that he could then correct if he is not happy with its behaviour.

To take into account those two points, a new learning model has to be defined. We present here the active learning model. The `SQUIRREL` system that we develop here is a learning algorithm for monadic node selection queries in this framework that allows the user—presented here as an oracle—to interact with a core learning algorithm, which will be a variant of `tRPNI` able to deal with partial annotations.

### 5.1. Active learning model

Deterministic finite automata are learnable in the MAT model (Angluin, 1987), i.e., by an active learner who may ask membership and equivalence queries to his teacher. This result extends to deterministic bottom-up tree automata (Sakakibara, 1990; Drewes & Hogberg, 2003). Such queries, however, are not well suited for active learning of node selection queries in trees. Indeed, a node selection query may be represented by a regular tree language over $\Sigma \times \texttt{Bool}$. Thus, a membership query would need as input an annotated tree which is not manageable in practice. Moreover, a membership query answers yes or no whether the annotated tree is correctly annotated, but, if the answer is negative, a membership query gives no information on the annotation error. Also, recall that a monadic query can be represented by several languages of annotated trees. Therefore equivalence of tree languages is not adapted for equivalence of monadic node selection queries.

We introduce two new types of queries that the learner may ask to the teacher, *correct labeling queries (CLQs)* and *query equivalence queries (QEQs)*. This can be done with respect to an arbitrary formalism for representing queries—for instance by NSTTs. We denote query representations by $A$ and the query they represent by $q_A$. A CLQ about target $q$ is defined as follows:

**Correct Labeling Query (CLQ)**

| | |
|---|---|
| **Input**: | an annotated $\Sigma \times \texttt{Bool}$-tree $t \times \beta$ |
| **Output**: | yes if $q(t) = \beta$, else some node $v \in \texttt{nodes}(t)$ such that $\beta(v) \neq q(t)(v)$ |

A CLQ asks whether the current annotation hypothesis $t \times \beta$ is correct and requires a failure witness $v$ otherwise. A QEQ about target $q$ has the form:

**Query Equivalence Query (QEQ)**

| | |
|---|---|
| **Input**: | a query representation $A$ |
| **Output**: | yes if $q = q_A$, else some $\Sigma$-tree $t$ s. t. $q_A(t) \neq q(t)$. |

A QEQ asks whether the current query hypothesis $A$ is correct, and requires a failure witness $t$ otherwise.

Before presenting the main result, let us discuss about queries in practice. The user (or query designer) plays the role of the oracle. This is not a problem for CLQ: the user can check on the current web page if it is well annotated and correct it otherwise. This may however be a problem for QEQ. We do not want to show the inferred NSTT to the user, as we want a "black-box" system usable even for non-expert users. The only thing that the user can check is whether the inferred query annotates correctly web pages that he proposes. The user simply answers "yes" to the QEQ (by actually stopping the learning process) when he is satisfied with the behavior of the proposed query on web pages processed so far. The possible consequence is that this query may not behave correctly on some unseen web pages, especially web pages that have a general design which is different to the one of processed web pages. Solutions to this problem are discussed in Section 6 and in the conclusion. Nevertheless QEQs and CLQs allow to define a rigorous framework for interactive learning of monadic queries. We have the following result:

**Theorem 2.** *Monadic node selection queries represented by deterministic* NSTT*s are learnable in polynomial time by active learning processes with correct labeling queries and query equivalence queries.*

**Proof:** Let $q$ be the target query and $L$ be a target functional language over $\Sigma \times \mathtt{Bool}$ defining $q$. Let $\mathcal{A}$ be a learning algorithm with membership and equivalence queries for deterministic tree automata. At each call of $\mathcal{A}$ to a membership query $t \times \beta \in L$, we ask the correct labeling oracle with input $t \times \beta$ and answer yes if the CLQ answers yes and no if the CLQ outputs a node. At each call of $\mathcal{A}$ to an equivalence query, we ask the query equivalence query oracle with the same hypothesis tree automaton and answers yes if the QEQ outputs yes. If the QEQ outputs a $\Sigma$-tree $t$, we have to construct the annotated tree $t \times \beta$ which is a counterexample for the equivalence query. To do so, ask the correct labeling oracle until we get the annotated tree $t \times \beta$. The reader should note that the number of calls to the correct labeling oracle is bounded by the number of nodes to be selected in $t$. The modified algorithm is a learning algorithm with CLQs and QEQs. It outputs in polynomial time a minimal tree automaton $A$ of the functional language $L$, i.e. a deterministic NSTT defining the target query $q$. □

### 5.2. The SQUIRREL algorithm

The outer loop of the SQUIRREL algorithm in Fig. 13 halts when the teacher considers the current query to be equivalent to the target query. Otherwise a new tree is investigated. The inner loop halts when the teacher confirms that the current tree is correctly annotated by the current hypothesis query. Otherwise, it asks for further partial annotations for the current tree and learns from them.

A *partially annotated tree* over $\Sigma$ is a triple $\langle t,\ p_+,\ p_-\rangle$ consisting of a $\Sigma$-tree $t$, a set $p_+ \subseteq \mathtt{nodes}(t)$ of *positively annotated* nodes, a set $p_- \subseteq \mathtt{nodes}(t)$ of *negatively annotated* nodes, and $p_+ \cap p_- = \emptyset$. It is *consistent* with a query $q$ if $p_+ \subseteq q(t)$ and $p_- \cap q(t) = \emptyset$.

SQUIRREL

// fix a representation formalism for queries //
// we write $q_A$ for the query represented by $A$ //
let $A$ represent the empty query
let $S$ be the empty set of completely annotated trees
**Loop** until QEQ(A) returns yes
  let $t$ be the output of QEQ
  let $p_+ = p_- = \emptyset$ be partial annotations for $t$
  **Loop** until CLQ($t \times q_A(t)$) returns yes
    let $v$ be output of CLQ
    // add the correct annotation of $v$ to $p_+$ or $p_-$ //
    **If** $v \in q_A(t)$ **then** add $v$ to $p_-$ **else** add $v$ to $p_+$
    // learn a new query representation//
    $A \leftarrow \mathtt{RPNI}_{\mathrm{prune}}(S, \langle t, p_+, p_- \rangle)$
  Add $t \times q_A(t)$ to the set $S$
**Output :** $A$

**Fig. 13** SQUIRREL active learning algorithm

SQUIRREL is parametrized by an arbitrary algorithm $\mathtt{RPNI}_{\mathrm{prune}}$, which we will fix later on to be a variant of the tRPNI algorithm for NSTTs. However, we could use every other algorithm that computes query representations from a set $S$ of (completely) annotated trees (the set of correctly annotated trees seen so far) and a partially annotated tree $\langle t, p_+, p_- \rangle$ (the tree for which the teacher is answering CLQ queries).

### 5.3. Pruning

In order to be able to benefit from partial annotations, we propose to prune subtrees that seem irrelevant for distinguishing positively annotated nodes. All remaining nodes are assumed to be unselected if not annotated positively (which seems reasonable for appropriate pruning heuristics). Thereby, we obtain complete annotations on pruned trees. Note that negatively annotated nodes may be pruned. This complicates the situation a little, as wrapper induction needs to ensure consistency with all negative annotations, pruned or not.

At the same time, pruning helps resolving the second drawback because the size of the target automaton is reduced. The idea is similar to that of windowing in wrapper induction from texts (see e.g. Freitag & Kushmerick, 2000; Chidlovskii, 2001).

### 5.4. RPNI with pruning

The learning algorithm $\mathtt{RPNI}_{\mathrm{prune}}$ in Fig. 14 is parametrized by a *pruning function* prune, which is an arbitrary function mapping $\Sigma \times \mathtt{Bool}$-trees to $(\Sigma \times \mathtt{Bool})_T$-trees such that for every possible argument $t$:

$$\mathtt{prune}(t) \in \mathtt{cuts}(t)$$

First, we prune all input trees in $S$ and the partially annotated input tree $\langle t, p_+, p_- \rangle$. This is done by computing a completely annotated tree $t \times p_+$ in which all nodes of $p_+$ are

**Fig. 14** Learning from partially annotated trees

$\texttt{RPNI}_{\text{prune}}$ $(S, \langle t, p_+, p_- \rangle)$

// prune all input trees //
$S' = \{\texttt{prune}(t' \times \beta) \mid t' \times \beta \in S\} \cup \texttt{prune}(t \times p_+)$
// compute a deterministic PNSTT by RPNI//
$A \leftarrow \text{PNSTT}(S')$
**For** $i = 1$ **to** $|\texttt{states}(A)|$ **do**
  **For** $j = 0$ **to** $i - 1$ **do**
    $A' \leftarrow \text{det-merge}(A, q_i, q_j)$
    **If** $A'$ is cut-functional
    **and** $A'$ consistent with $S$ and $\langle t, p_+, p_- \rangle$
    **then** $A \leftarrow A'$
**Output :** $A$

labeled by 1 and all others by 0, to which prune is then applied. Let $S'$ be the resulting set of completely annotated pruned trees.

Second, $\texttt{RPNI}_{\text{prune}}$ infers a PNSTT by a variant of the RPNI algorithm which tests for cut-functionality. It starts with an initial PNSTT for $S'$, denoted by $\text{PNSTT}(S')$, as usual with RPNI. This is the deterministic PNSTT whose language consists of all pruned trees in $S'$, such that for all states $q \in \texttt{states}(\text{PNSTT}(S'))$ there exists exactly one tree $t$ over $(\Sigma \times \texttt{Bool})_T$ with a successful run by $A$. Then, we use a state merging for generalization. The function det-merge takes as input an automaton $A$ and two states $q_i$ and $q_j$ of $\texttt{states}(A)$ and outputs an automaton where $q_i$ and $q_j$ are merged such that det-merge$(A, q_i, q_j)$ is made deterministic.

The iteration continues with the merged automaton if it is cut-functional and consistent with all negative annotations in $S$ and $\langle t, p_+, p_- \rangle$, or else refuses the merging step and continues with the previous automaton. *Consistency* here means that there does not exist any tree $t' \times \beta \in S$ and node $v \in q_A(t')$ such that $\beta(v) = 0$, and there does not exist any node $v \in p_-$ such that $v \in q_A(t)$.

*Improvements.* The most classical one is to tweak the merging order, as illustrated by evidence driven state merging (Lang, Pearlmutter, & Price, 1998). Here, we choose a merging order which favors merges leading to PNSTTs with "horizontal" recursion, since this type of recursion is more frequent in semi-structured documents. The second improvement is to use background knowledge in form of typing automata (Kermorvant & de la Higuera, 2002). For web documents, we use a very general typing automaton that recognizes all well-formed web documents. Merges between states of different types are forbidden. This restricts the class of PNSTTs that can be inferred.

To conclude, the reader should note that $\texttt{RPNI}_{\text{prune}}$ responds in polynomial time (as tRPNI and RPNI). As the cut-functionality of a tree automaton $A$ can be tested in $O(|A|^3)$ (see Proposition 2), which is the most complex task of the inner loop, and as this test has to be performed a quadratic amount of time, the overall complexity of tRPNI is $O(|S|^5)$ (at worst, the working automaton is always the initial automaton, which has a size at worst equal to the size of the input sample). Note that we consider here the size of an automaton as the sum of its number of node and its number of rules, and the size of a sample as the total number of nodes of trees in the sample. For large trees and a strongly pruning function there would be a speed-up of the learning process. However there is a trade-off in that we loose completeness in learnability. We discuss this last point in the next subsection.

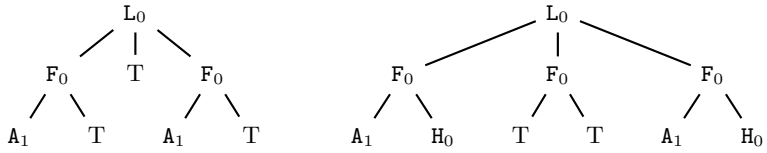



**Fig. 15** The pruned trees applying the strategy `paths_only` (left) and `paths_extended` (right) to the tree `films`

### 5.5. Pruning heuristics

The choice of the pruning function determines the quality of $\texttt{RPNI}_{\text{prune}}$. To be able to learn a target query, some conditions should be imposed on the pruning function. Namely that it should be possible to find a characteristic pruned set for the target language, given the pruning function. And that it should be possible to find a cut-functional pruned language satisfying the target, for a given pruning function. Therefore the class of learnable functions depends on the pruning function. The time complexity of the learning algorithm is measured w.r.t. the size of pruned trees. Thus, the more aggressive the pruning function is, the more efficient the learning algorithm will be. Generally, we will only consider pruning functions that keep all positively annotated nodes. We have considered three pruning strategies:

- The function `paths_only` cuts all nodes except those that are ancestors of or equal to positively labeled nodes. An example is given in Fig. 15. Note that more than a single path is kept so that induction is not simply reduced to learning path expressions. The problem of defining the class of NSTTs that can actually be identified by $\texttt{RPNI}_{\text{prune}}$ with the pruning function `paths_only` is still open. The experiments show that pruning by `paths_only` yields significantly better learning efficiency than less aggressive pruning functions. Also experimental results show that expressiveness is sufficient for monadic queries based on the structure.
- The function `paths_extended` additionally keeps all the nodes whose parents are on such paths. An example is given in Fig. 15. The problem of defining the class of NSTT that can actually be identified by $\texttt{RPNI}_{\text{prune}}$ with the pruning function `paths_extended` is still open. The class is larger than for the pruning function `paths_only` because some node selection queries using textual contents are learnable. For instance, a query for actor lists where the next sibling contains the textual value "Hitchcock". As soon as textual values are not involved, experimental results have shown lower learning efficiency and equivalent expressiveness than for the pruning function `paths_only`.
- The function `identity`. $\texttt{RPNI}_{\text{prune}}$ restricted to completely annotated trees behaves as the `tRPNI` algorithm for deterministic NSTTs. All MSO-definable node selection queries could be the target of $\texttt{RPNI}_{\text{prune}}$. However experimental results show that a large number of annotated examples is required to achieve learning. Also, in the interactive setting, learning from a small number of annotated positions could introduce wrong generalizations.

## 6. Squirrel Web information extraction system

We have developed a software tool, `SQUIRREL` (Fig. 16). We first describe the system. Then we give experimental results in the off-line setting in order to compare our system with others.

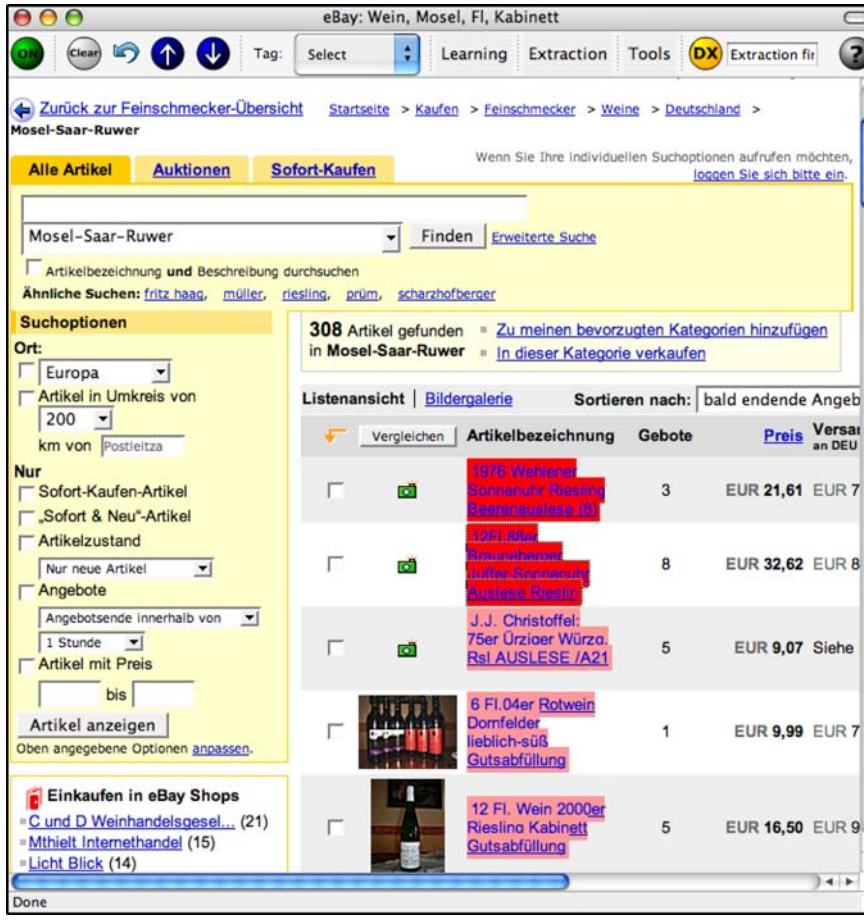



**Fig. 16** Screenshot of Mozilla Firefox with SQUIRREL: a toolbar has been added to help the user to annotate data in the web page and to dialog with the learning algorithm

Finally we give experimental results in order to evaluate the number of interactions in the interactive setting.

*System description.* SQUIRREL preprocesses pages via the parser included in Mozilla. This yields a DOM XHTML tree. We ignored textual values. Attributes were ignored too except for ID and CLASS, which were merged with the name of the tag, i.e. <DIV ID = title> gives a node labeled DIVtitle.

The user is the query designer. When a CLQ is made by SQUIRREL, the query designer annotates an incorrect node (unwanted or missing). In a real setting, the query designer may annotate more than one position. When a QEQ is made by SQUIRREL, the query designer decides to halt the induction process or chooses a new Web page.

The SQUIRREL algorithm calls the $\text{RPNI}_{\text{prune}}$ algorithm. For the experimental results we have chosen the pruning function paths_only.

*Off-line experiments.* The aim of these experiments is to compare $\texttt{RPNI}_{\text{prune}}$ with state-of-the-art algorithms. Thus we do not consider the interactive framework and $\texttt{RPNI}_{\text{prune}}$ is given as an input completely annotated examples for the target query.

We considered benchmarks from RISE[2], the only public repository of annotated documents designed to evaluate information extraction algorithms on Web pages. We had to discard some of the datasets, because the extraction tasks they defined were not HTML element extraction tasks, and could therefore not be addressed by our algorithm. Typically, extracting a given word in a sentence is a problem outside the scope of our work.

The remaining datasets we used are these ones:

- 18 of Kushmerick's WIEN benchmark (Freitag & Kushmerick, 2000) used by Muslea in Muslea, Minton, & Knoblock (2001) to establish a comparison between Wien and Stalker. They are denoted by S-*XX* in table of results. Each of them is constituted with 8 pages, and is associated with one to four extraction tasks.
- Bigbook, a set 235 pages extracted from an address book. Two tasks are associated with this set: name extraction and address extraction.
- Okra, a set of 250 pages which corresponds to a people search engine output. Three tasks are associated with this set: name, mail address and score extraction.

For each dataset, we evaluate our algorithm using a 8-fold cross-validation protocol. The dataset is divided in 8 parts. A wrapper dedicated to one extraction task is learned from 7 parts, and tested on the last one. True positives, false negatives and false positives are counted and a F1-measure score is computed. Each part of the dataset is used as test set for each extraction task in a distinct experience, and an average F-measure is computed.

It is rather difficult to compare our own results to the ones of other systems, considering that we cannot test them ourselves and that we have to use published results whose experimental protocol are never exactly the same. Following Muslea in Muslea, Minton, and Knoblock (2001), we establish a comparison using a qualitative criteria: we consider three possible behaviours for each system on each benchmark: perfect (average F-measure of 100%), imperfect (average F-measure between 90% and 100%) and failure (F-measure below 90%). All results are presented in Fig. 17. Results concerning WIEN (Kushmerick, 2000) and Stalker (Muslea, Minton, & Knoblock, 2001), two well known information extraction systems, are given for the sake of comparison. They have been directly taken from Muslea, Minton, and Knoblock (2001).

These results show that `SQUIRREL` behaves satisfyingly in a large majority of the benchmarks, and achieves performances on these benchmarks which are equals or greater than Wien and Stalker. A closer analysis of the failures of Squirrel (S-6 and S-11) shows that they have expected causes: in both cases, the structure is not informative enough because textual contents of leaves are necessary to define the query. Therefore `SQUIRREL` with the pruning function `paths_only` can not infer a relevant wrapper. One way to overcome this problem is to preprocess the data sets in order to distinguish text nodes which should be useful for the induction process, consider trees with these textual values, and use the pruning function `paths_extended`.

To end the comparisons with existing systems, we now compare our system with the system presented in Raeymaekers, Bruynooghe, and Van den Bussche (2005). They represent a monadic node selection query $q$ by a tree language $L_q$ where each tree contains a single marked node. To decide if a node of a tree $t$ is to be selected, the node is marked (replaced

[2] http://www.isi.edu/~muslea/RISE/index.html.

| | Stalker | Wien | **Squirrel** |
|---|---|---|---|
| S-4 | √ | √ | √ |
| S-5 | √ | √ | √ |
| S-6 | ≃ | – | – |
| S-7 | √ | – | √ |
| S-8 | √ | √ | √ |
| S-9 | ≃ | – | √ |
| S-10 | √ | √ | √ |
| S-11 | ≃ | – | – |
| S-12 | √ | √ | √ |
| S-13 | √ | √ | √ |
| S-14 | √ | √ | √ |
| S-19 | √ | √ | √ |
| S-20 | √ | √ | √ |
| S-22 | √ | √ | √ |
| S-23 | √ | √ | √ |
| S-24 | ≃ | – | √ |
| S-25 | √ | √ | √ |
| S-28 | √ | √ | √ |
| S-30 | √ | √ | √ |
| Okra | √ | √ | √ |
| BigBook | √ | √ | √ |

**Fig. 17** Comparison between Stalker, Wien and Squirrel on RISE corpus. The symbol—means *failure*, that is average F-measure lower than 90%, ≃ means *imperfect*, that is average F-measure between 90% and 100%, and √ means *perfect*, that is average F-measure equal to 100%

by a special symbol), if the tree is in $L_q$, then the original node is to be selected. Whereas we use the `RPNI` algorithm from positive and negative examples, their base learning algorithm is from positive examples only. They consider a subclass of the class of regular languages which is chosen to be the class of $(k, l)$-contextual tree languages where $k$ and $l$ are parameters. With parameter tuning, they achieve perfect results on benchmarks presented Fig. 17. They succeed in S-6 and S-11 because they use a preprocessing procedure to distinguish important textual values. They also present a method to learn the parameters from positive and negative examples.

*Benchmarking interactive learning.* For the experiments, we used the benchmarks Okra, Bigbook and real datasets[3] from different Web sites:

- the `Yahoo` dataset contains 80 pages of the Yahoo Web directory. The task is to extract a set of URLs from a list of URLs. There are different lists on a page, the pages have different structures and the position of the list is not fixed;
- the `eBay` dataset contains 40 pages of products. The task is to extract product names from a table. The product name can be mixed with other information in the cell;
- the `Google` dataset contains 40 results of the search engine. The task is to extract the links. The structure of pages is not fixed and there are numerous auxiliary links;
- the `New York Times` dataset contains 40 article pages with different structures. The task is to extract texts of the articles.

In order to evaluate the efficiency of the algorithm in the interactive mode, we simulated the behavior of a query designer. `SQUIRREL` is interfaced with a user simulator $S$: at each call of the correct labeling oracle, $S$ answers the first wrong annotation of the page in reading

[3] http://www.grappa.univ-lille3.fr/~carme/corpus.html

| | #QEQ | #CLQ |
|---|---|---|
| Okra-names | 1.6 | 3.48 |
| Bigbook-addresses | 1 | 3.02 |
| Yahoo | 6.18 | 11.36 |
| E-bay | 1.06 | 2.62 |
| NYTimes | 1.44 | 1.44 |
| Google | 1.86 | 4.78 |

**Fig. 18** $\#QEQ$ denotes the number of equivalence queries and $\#CLQ$ denotes the number of correct labeling queries. Results are averaged over 100 experiments

order of the source file (which is the first positive annotation of the page for the first call) ; at each call of the equivalence oracle, $S$ returns a Web page on which the pruned NSTT is not correct.

In an interactive setting, time complexity is important. For the different data sets, on a standard computer, the average time for learning the current hypothesis and annotating the current Web page is between 1 and 2 seconds and less than 5 seconds on the more intricate cases.

We also measured the number of CLQs and QEQs which are needed to achieve perfect results on the set of Web pages. The results are given in Fig. 18. The number of CLQs is the number of corrections made by the query designer. The results show that this number is quite small. We should note that this number is over-valued because in a real setting, the query designer may correct more than one wrong annotation at a time, and we can hope that his choice of correction is more relevant than the one used in our protocol. The number of QEQs is the number of Web pages necessary to achieve perfect results. The number of necessary Web pages for the `Yahoo` dataset is greater than for other datasets because the dataset is heterogeneous. We should note that the number of necessary Web pages is lower than the number of Web pages viewed by the query designer. Indeed, in our protocol the user simulator selects only Web pages on which the current hypothesis is incorrect while, in the real setting, the query designer may choose a Web page on which the current hypothesis is correct. Therefore we now discuss the choice of the next Web page to be processed in the interactive setting.

*Towards active learning.* We should render our model active in the sense that the learner intelligently proposes the next Web page among a set of Web pages—the Web pages of the target Web site.

We considered a method estimating the value of a tree according to the number of extracted nodes by the current hypothesis wrapper. We supposed that there is a linear dependence between the number of extracted nodes and the total number of nodes. This lead to the following strategy: let $L$ be the set of Web pages not seen so far by `SQUIRREL`; for every Web page $p$ in $L$, we compute the number $x$ of nodes of $p$ and $y$ the number of selected nodes by the current hypothesis wrapper; we used the test of Grubbs for outlier selection to select the web page (the farthest point $(x, y)$ from the regression line). The selected Web page is processed by the main loop of `SQUIRREL`.

We only considered the three most difficult datasets. We computed performance with a learning set of annotated documents of respectively 1, 2, 5, 10 Web pages. These Web pages were either randomly chosen or chosen according to the strategy given above. The experimental results are given in Fig. 19. They show that our naive strategy for active learning leads to significant improvements. It remains to find an intelligent active learning strategy as a future work.

**Fig. 19** F-measure of inferred query on test set, using 1, 2 5 or 10 documents chosen randomly (top) or chosen accordingly to our strategy (bottom)

| | 1 | 2 | 5 | 10 |
|---|---|---|---|---|
| | random | | | |
| Yahoo | 71.8 | 80.9 | 82.1 | 93.7 |
| NYTimes | 91.2 | 92.4 | 95.3 | 100 |
| Google | 98.7 | 99.1 | 99.5 | 99.8 |
| | active | | | |
| Yahoo | 71.8 | 86.1 | 98.4 | 99.4 |
| NYTimes | 91.2 | 100 | 100 | 100 |
| Google | 98.7 | 100 | 100 | 100 |

## 7. Conclusion

We have presented a wrapper induction system with the following characteristics:

- it generates tree-based wrappers which select nodes in trees;
- it uses new machine learning techniques based on grammatical inference and active learning;
- it is integrated in a visual environment;
- it has simple interactions with the query designer.

From the theoretical point of view, it is proved that NSTTs and PNSTTs can express all MSO-definable node selection queries. However any pruning function, different from identity, will restrict the class of learnable node selection queries. In future work, We have to specify the subclass of node selection queries which is learnable depending on the chosen pruning strategy.

From the practical point of view, we have proved that our system achieves state-of-the-art performance on standard benchmarks and that the number of interactions with the query designer is quite small. Nevertheless there are still a number of possible improvements. First, we should render our algorithm active with intelligent strategies. Second, we do not consider textual values and omit some attribute values. Therefore, we should adapt our approach to take textual information into account. Finally, our current system is limited to one-slot information extraction since we only consider monadic node selection queries over trees. Work is under progress to extend our approach to multi-slot information extraction, i.e., the problem of learning $n$-ary regular queries in trees.

## A. Testing cut-functionality in polynomial time

We present a direct proof for Proposition 2 in the case of binary trees (and thus for unranked trees), stating that cut-functionality of deterministic tree automata can be decided in polynomial time.

We characterize cut-functionality of a tree automaton $A$ with signature $(\Sigma \times \mathsf{Bool})_T$ by a binary relation $\mathtt{inc}_A \subseteq \mathtt{states}(A)^2$, testing for two states whether they can be reached by two runs of $A$ on some incompatibly annotated compatible trees. For all two states $p, p' \in \mathtt{states}(A)$:

$$\mathtt{inc}_A(p, p') \text{ iff } \begin{cases} \exists t, t' \text{ compatible trees over } \Sigma_T \\ \exists \beta, \beta' \text{ incompatible trees over } \mathsf{Bool}_T \\ \exists r \in \mathtt{runs}_A(t \times \beta) : \ r(\varepsilon) = p \\ \exists r' \in \mathtt{runs}_A(t' \times \beta') : \ r'(\varepsilon) = p' \end{cases}$$

**Lemma 4.** *A tree automaton $A$ over $(\Sigma \times \texttt{Bool})_T$ is cut-functional if and only if $\texttt{inc}_A(p, p')$ doesn't hold for any two states $p, p' \in \texttt{final}(A)$.*

**Proof:** Suppose $\texttt{inc}_A(p, p')$ holds for some $p, p' \in \texttt{final}(A)$. Then there exist runs $r \in \texttt{runs}_A(t \times \beta)$ and $r' \in \texttt{runs}_A(t' \times \beta')$ for incompatible trees $\beta$ and $\beta'$ and compatible trees $t$ and $t'$. These runs are successful since $p, p' \in \texttt{final}(A)$. Thus, $A$ is not cut-functional. The argument can be reversed. □

It remains to show that the relation $\texttt{inc}_A$ can be computed in polynomial time. We prove that it can be computed by a saturation procedure over states. We first need to define another binary relation $\texttt{sim}_A$ on states. $\texttt{sim}_A$ holds if two states can be reached by compatible annotated trees:

*Definition 8.* Let $A$ be a deterministic tree automaton over $(\Sigma \times \texttt{Bool})_T$, $p$ and $p'$ two states of $A$, then $\texttt{sim}_A(p, p')$ holds if there exist two compatible $(\Sigma \times \texttt{Bool})_T$-trees $t \times \beta, t' \times \beta'$, a run $r$ of $A$ over $t \times \beta$ such that $r(\varepsilon) = p$, and a run $r'$ of $A$ over $t' \times \beta'$ such that $r'(\varepsilon) = p'$.

We now give an inductive characterization of the relation $\texttt{inc}_A$ which is based on the relation $\texttt{sim}_A$ and on the rule set of the tree automaton. We call an automaton *trimmed* if it has no useless state, i.e., if for each state $p$ there is a run $r$ of $A$ on some tree such that $r(\varepsilon) = p$.

**Lemma 5.** *Let $A$ be a trimmed deterministic tree automaton over $(\Sigma \times \texttt{Bool})_T$, $\texttt{inc}_A(p, p')$ holds if and only if there exist rules satisfying one of the cases in Fig. 20*

**Proof:** Let us suppose that $\texttt{inc}_A(p, p')$. Then there exist two $(\Sigma \times \texttt{Bool})_T$-trees $t \times \beta, t' \times \beta'$ such that $t$ and $t'$ are compatible but $t \times \beta$ and $t' \times \beta'$ are not compatible, a run $r$ of $A$ over $t \times \beta$ such that $r(\varepsilon) = p$, and a run $r'$ of $A$ over $t' \times \beta'$ such that $r'(\varepsilon) = p'$.

- First, note that neither $t$ nor $t'$ is equal to $T$ (which would correspond to a tree entirely pruned), as otherwise $\beta = T$ (resp. $\beta' = T$), and as $T$ is compatible with every tree, we would have $\beta$ and $\beta'$ compatible.
- If $t$ is a single leaf tree with $t = a$ then $t' = a$ also as $t$ and $t'$ are compatible. In this case $\beta = b$ and $\beta' = \neg b$ as they are not compatible. As $r(\varepsilon) = p$ and $r'(\varepsilon) = p'$, this implies that there are two rules $(a, b) \to p$ and $(a, \neg b) \to p'$ and (i) holds.
- If $t$ is a tree of the form $t = a(t_1, t_2)$ then, as $t$ and $t'$ are compatible, there are trees $t'_1$ and $t'_2$ such that $t' = a(t'_1, t'_2)$ such that $t_1$ is compatible with $t'_1$ and $t_2$ is compatible

$$
\begin{array}{ll}
(i) & (a,b) \to p \in \texttt{rules}(A) \\
& \quad \wedge\ (a,\neg b) \to p' \in \texttt{rules}(A) \\
(ii) & (a,b)(p_1,p_2) \to p \in \texttt{rules}(A) \\
& \quad \wedge\ (a,\neg b)(p'_1,p'_2) \to p' \in \texttt{rules}(A) \\
& \quad \wedge\ \texttt{sim}_A(p_1,p'_1) \wedge \texttt{sim}_A(p_2,p'_2) \\
(iii) & (a,b)(p_1,p_2) \to p \in \texttt{rules}(A) \\
& \quad \wedge\ (a,b')(p'_1,p'_2) \to p' \in \texttt{rules}(A) \\
& \quad \wedge\ \texttt{inc}_A(p_1,p'_1) \wedge \texttt{sim}_A(p_2,p'_2)
\end{array}
\qquad
\begin{array}{ll}
(iv) & (a,b)(p_1,p_2) \to p \in \texttt{rules}(A) \\
& \quad \wedge\ (a,b')(p'_1,p'_2) \to p' \in \texttt{rules}(A) \\
& \quad \wedge\ \texttt{sim}_A(p_1,p'_1) \wedge \texttt{inc}_A(p_2,p'_2) \\
(v) & (a,b)(p_1,p_2) \to p \in \texttt{rules}(A) \\
& \quad \wedge\ (a,b')(p'_1,p'_2) \to p' \in \texttt{rules}(A) \\
& \quad \wedge\ \texttt{inc}_A(p_1,p'_1) \wedge \texttt{inc}_A(p_2,p'_2)
\end{array}
$$

**Fig. 20** Cases of Lemma 5

with $t'_2$. Let $t \times \beta = (a, b)(t_1 \times \beta_1, t_2 \times \beta_2)$ and $t' \times \beta' = (a, b')(t'_1 \times \beta'_1, t'_2 \times \beta'_2)$. Let us consider the rules $(a, b)(p_1, p_2) \to p$ and $(a, b')(p'_1, p'_2) \to p'$ which are applied at the root of the trees $t \times \beta$ and $t' \times \beta'$ in the runs $r$ and $r'$. As $t_1$ and $t'_1$ are compatible, and $t_2$ and $t'_2$ are compatible, the states satisfy $\mathtt{sim}_A(p_1, p'_1)$ or $\mathtt{inc}_A(p_1, p'_1)$ and $\mathtt{sim}_A(p_2, p'_2)$ or $\mathtt{inc}_A(p_2, p'_2)$. First, consider the case where the Boolean values at the root of the trees $t \times \beta$ and $t' \times \beta'$ are different. In this case, one of the four conditions (ii), (iii), (iv) or (v) holds. If the Boolean values at the root of the trees $t \times \beta$ and $t' \times \beta'$ are equal, then $\mathtt{inc}_A(p_1, p'_1)$ or $\mathtt{inc}_A(p_2, p'_2)$ must hold and one of the conditions (iii), (iv) or (v) holds.

We now prove the other direction of the equivalence.

- (i) If $((a, b) \to p \in \mathtt{rules}(A) \wedge (a, \neg b) \to p' \in \mathtt{rules}(A))$ then $\mathtt{inc}_A(p, p')$.
- (ii) Let us consider two rules $(a, b)(p_1, p_2) \to p$ and $(a, \neg b)(p'_1, p'_2) \to p'$ in $\mathtt{rules}(A)$ such that $\mathtt{sim}_A(p_1, p'_1)$ and $\mathtt{sim}_A(p_2, p'_2))$. Then $\mathtt{sim}_A(p_1, p'_1)$ implies that there are two compatible trees $t_1 \times \beta_1$ and $t'_1 \times \beta'_1$, and runs $r_1 \in \mathtt{runs}_A(t_1 \times \beta_1)$ and $r'_1 \in \mathtt{runs}_A(t'_1 \times \beta'_1)$ such that $p_1 = r_1(\varepsilon)$ and $p'_1 = r'_1(\varepsilon)$. We have the same property for the right sub-term (the corresponding states, runs and terms are denoted by $p_2$, $p'_2$, $r_2$, $r'_2$, $t_2 \times \beta_2$ and $t'_2 \times \beta'_2$). Therefore we can consider trees $t \times \beta = (a, b)(t_1 \times \beta_1, t_2 \times \beta_2)$ and $t' \times \beta' = (a, \neg b)(t'_1 \times \beta'_1, t'_2 \times \beta'_2)$ associated to $p$ and $p'$ through runs $r = p(r_1, r_2)$ and $r' = p'(r'_1, r'_2)$. $t \times \beta$ and $t' \times \beta'$ are not compatible whereas $t$ and $t'$ are, therefore $\mathtt{inc}_A(p, p')$ holds.
- (iii) We consider two rules satisfying (iii). We consider the two trees that are associated with the property $\mathtt{inc}_A(p_1, p'_1)$, and then we construct two trees proving that $\mathtt{inc}_A(p, p')$ holds. The construction is similar to the one used in the precedent point. In this case, incompatibility between Boolean values comes from the left subtrees $t_1 \times \beta_1$ and $t'_1 \times \beta'_1$ because of $\mathtt{inc}_A(p_1, p'_1)$.
- the cases (iv) and (v) are similar.

□

**Lemma 6.** *Let $A$ be a trimmed deterministic tree automaton over $(\Sigma \times \mathtt{Bool})_T$, $\mathtt{sim}_A(p, p')$ stands if and only if there exist rules satisfying one of the following cases:*

$$
\begin{array}{ll}
(i) & (p = p') \\
(ii) & (T \to p) \in \mathtt{rules}(A) \\
(iii) & (T \to p') \in \mathtt{rules}(A) \\
(iv) & \exists((a, b)(p_1, p_2) \to p \in \mathtt{rules}(A) \\
 & \quad \wedge (a, b)(p'_1, p'_2) \to p' \in \mathtt{rules}(A) \\
 & \quad \wedge (\mathtt{sim}_A(p_1, p'_1)) \wedge (\mathtt{sim}_A(p_2, p'_2))
\end{array}
$$

**Proof:** If $\mathtt{sim}_A(p, p')$, then there exist two compatible $(\Sigma \times \mathtt{Bool})_T$-trees $t \times \beta$ and $t' \times \beta'$, a run $r$ of $A$ over $t \times \beta$ such that $r(\varepsilon) = p$, and a run $r'$ of $A$ over $t' \times \beta'$ such that $r'(\varepsilon) = p'$.

- If $t = T$, then there is a rule $T \to p$ in $A$ and $(ii)$ holds.
- If $t \times \beta = (a, b)$, then either $t' \times \beta' = (a, b)$ or $t' \times \beta' = (T, T)$ because $t \times \beta$ and $t' \times \beta'$ are compatible. In the first case, because of determinism, $p = p'$ and $(i)$ holds. In the second case, there is a rule $T \to p'$ in $A$ and $(iii)$ holds.
- If there are some trees $t_1 \times \beta_1$ and $t_2 \times \beta_2$ such that $t \times \beta = (a, b)(t_1 \times \beta_1, t_2 \times \beta_2)$, then either $t'$ is equal to $T$ (and $(iii)$ applies) or the root of $t'$ is also $(a, b)$ as $t \times \beta$ and $t' \times \beta'$ are compatible. Let $t' \times \beta' = (a, b)(t'_1 \times \beta'_1, t'_2 \times \beta'_2)$. Let us consider the rules $(a, b)(p_1, p_2) \to p$ and $(a, b)(p'_1, p'_2) \to p'$ which are applied at the root of the trees $t \times \beta$

and $t' \times \beta'$ in the runs $r$ and $r'$. We should note that $t_1 \times \beta_1$ and $t'_1 \times \beta'_1$ are compatible, and $t_2 \times \beta_2$ and $t'_2 \times \beta'_2$ are compatible, thus the states satisfy $\mathtt{sim}_A(p_1, p'_1)$ and $\mathtt{sim}_A(p_2, p'_2)$, and $(iv)$ holds.

We now prove the other direction of the equivalence.

- (i) If $p = p'$, let $t \times \beta$ be any tree that has a run in $A$ whose root is $p$, $t \times \beta$ is compatible with itself and $\mathtt{sim}_A(p, p')$.
- (ii) and (iii) If $T \to p \in \mathtt{rules}(A)$, then we can consider any tree $t' \times \beta'$ that has a run in $A$ whose root is $p'$ then $t' \times \beta'$ is compatible with $T$ and $\mathtt{sim}_A(p, p')$. Case $(iii)$ is similar.
- (iv) If $(a, b)(p_1, p_2) \to p$ and $(a, b)(p'_1, p'_2) \to p'$ in $\mathtt{rules}(A)$ with $\mathtt{sim}_A(p_1, p'_1)$ and $\mathtt{sim}_A(p_2, p'_2)$. The two relations $\mathtt{sim}_A(p_1, p'_1)$ and $\mathtt{sim}_A(p_2, p'_2)$ imply that there exist trees $t_1 \times \beta_1$, $t'_1 \times \beta'_1$, $t_2 \times \beta_2$ and $t'_2 \times \beta'_2$ that are associated to states $p_1$, $p'_1$, $p_2$ and $p'_2$ through runs $r_1$, $r'_1$, $r_2$ and $r'_2$ on $A$. Let $t \times \beta = (a, b)(t_1 \times \beta_1, t_2 \times \beta_2)$ and $t' \times \beta' = (a, b)(t'_1 \times \beta'_1, t'_2 \times \beta'_2)$, then $t \times \beta$ and $t' \times \beta'$ are compatible, $t \times \beta$ has a run $p(r_1, r_2)$ in $A$ that uses the rule $(a, b)(p_1, p_2) \to p$ and $t' \times \beta'$ has a run $p'(r'_1, r'_2)$ in $A$ that uses the rule $(a, b)(p'_1, p'_2) \to p'$. Therefore $\mathtt{sim}_A(p, p')$ holds. □

**Proposition 2.** *Cut-functionality of a deterministic tree automaton $A$ over $(\Sigma \times \mathtt{Bool})_T$ can be tested in cubic time.*

**Proof:** First, the relation $\mathtt{sim}_A$ is computed by a saturation process: Initialize with the first three rules in Lemma 6: consider the set of $\{(p, p) \mid p \in \mathtt{states}(A)\}$ union $\{(p, p') \mid \exists(T \to p \in \mathtt{rules}(A))\}$ union $\{(p, p') \mid \exists(T \to p' \in \mathtt{rules}(A))\}$; and then use the fourth rule (iv) inductively. There is at most a quadratic amount of linear tests to perform, therefore the construction is in cubic time. We use a similar construction for the relation $\mathtt{inc}_A$: First use the rule (i) in Lemma 5, second the rule (ii), and then apply inductively the rules (iii), (iv) and (v). Again, the construction is in cubic time. □

## B. Efficient query answering for NSTTs and pNSTTs

We next present algorithms for answering queries defined by NSTTs or pNSTTs. Query answering will be needed in our interactive learning environment for testing the current query hypothesis on given trees.

### B.1. Alternative definitions of monadic queries

We present two further ways for expressing monadic queries by tree automata. They will be useful for query answering for NSTTs and pNSTTs.

Every tree automaton $A$ with signature $\Sigma \times \mathtt{Bool}$ defines a MSO-definable monadic query $q_A$ in $\Sigma$-trees such that for all such trees $t$:

$$q_A(t) = \{v \in \mathtt{nodes}(t) \mid \beta(v) = 1 \text{ and } t \times \beta \in L(A)\}$$

We can compute $q_A(t)$ for a given binary tree $t$ and automaton $A$ by enumerating all annotated trees $t \times \beta$, filtering those that belong to $L(A)$, and collecting the nodes that such $\beta$'s annotates by 1. This way of answering queries defined by tree automata is highly non-deterministic, and thus inefficient. But we can do better, as we will see.

*Run-based automata queries.* offer another way to define monadic queries, for which efficient algorithms are known. They are defined by a tree automaton $A$ over $\Sigma$ and a set of selection states $S \subseteq \texttt{states}(A)$, and satisfy for all binary trees $t$:

$$q_{A,S}(t) = \{v \in \texttt{nodes}(t) \mid \exists r \in \texttt{succ\_runs}_A(t),\ r(v) \in S\}$$

This query selects all those nodes of $t$ are labeled by some selection state in $S$ in some successful run of $B$.

**Proposition 3** (Classical result). *Given a tree automaton $A$ over $\Sigma$, a set $S \subseteq \texttt{states}(A)$, and a binary $\Sigma$-tree $t$ one can compute the set $q_A(t)$ in time $O(|A| * |t|)$.*

The algorithm proceeds in two phases. In a bottom-up phase, every node $v$ of $t$ is annotated by all states of $B$ into which the subtree rooted by $v$ can be evaluated (determinization on the fly). In a second bottom-up phase, one underlines all those states that are used by successful runs. Finally, one selects all those nodes that are annotated by underlined states from the selection set $S$.

B.2. NSTTs and PNSTTs

We first consider monadic queries defined by tree automata $A$ over $\Sigma \times \texttt{Bool}$. We show that we can express them by run-based queries via automata projection (which will be useful later on too).

**Lemma 7.** *Queries by an automata $A$ over $\Sigma \times \texttt{Bool}$ can be defined equivalently by runs of the $\Sigma$ projected automaton $\pi(A)$: $q_A = q_{\pi(A),\texttt{states}(A)\times\{1\}}$.*

As a side product, Lemmas 7 show that every NSTT-defined query can be defined as run-based query by some unambiguous tree automaton.

**Proposition 4.** *Given a tree automaton $A$ over $\Sigma \times \texttt{Bool}$ and a binary $\Sigma$-tree $t$ one can compute the set $q_A(t)$ in time $O(|A| * |t|)$.*

By Lemma 7 we can convert the query $q_A$ into a run-based query by automata projection, which requires linear time. This later query can be answered in linear time by Proposition 3.

**Theorem 3.** *Queries $q_A(t)$ by* NSTT*s or* PNSTT*s over binary trees can be computed in linear time $O(|A| * |t|)$.*

For NSTTs this follows immediately from Proposition 4. Queries by PNSTTs can be answered the same way after conversion by the following lemma.

**Lemma 8.** *Every* PNSTT*s can be transformed in linear time into an automata over $\Sigma \times \texttt{Bool}$ that defines the same query.*

Note that the resulting automaton will not necessarily be an NSTT.

**Proof:** Let $A$ be a PNSTT with signature $(\Sigma \times \texttt{Bool})^T$. The idea of the transformation is to compute an automaton $\texttt{e}(A)$ over $\Sigma \times \texttt{Bool}$ that recognizes all expansions of trees in $L(A)$,

$$\frac{l \to p \in \mathtt{rules}(A) \qquad l \neq T}{l \to p \in \mathtt{rules}(\mathtt{e}(A))} \qquad \frac{a \in \Sigma}{(a,0) \to T \in \mathtt{rules}(\mathtt{e}(A))}$$

$$\frac{T \to p \in \mathtt{rules}(A)}{T \xrightarrow{\varepsilon} p \in \mathtt{rules}(\mathtt{e}(A))} \qquad \frac{a \in \Sigma}{(a,0)(T,T) \to T \in \mathtt{rules}(\mathtt{e}(A))}$$

**Fig. 21** Rules of automata $\mathtt{e}(A)$ that recognizes expansions of trees in $L(A)$

obtained by replacing $T$-leaves by arbitrary trees over $\Sigma \times \{0\}$.

$$\mathtt{states}(\mathtt{e}(A)) = \mathtt{states}(A) \uplus \{T\}$$
$$\mathtt{final}(\mathtt{e}(A)) = \mathtt{final}(A)$$

The rules of $\mathtt{e}(A)$ are induced by the schemata in 21, which are valid for all symbols $a \in \Sigma$, $p \in \mathtt{states}(A)$, and left hand sides $l$ of rules in $A$: all rules of $A$ are preserved by $\mathtt{e}(A)$ except those with the symbol $T$, which is moved from the signature into the states. Automaton $\mathtt{e}(A)$ may label all leaves by state $T$, as well as inner nodes whose descendants it has labeled by $T$ before. At an arbitrary time point, $\mathtt{e}(A)$ can decide to move from state $T$ to some state $p$ of $A$ for which $T \to p$ is a rule of $A$. Subsequently, $\mathtt{e}(A)$ behaves equally to $A$.

Note that we freely permit $\varepsilon$-rules, which can easily be eliminated while preserving the language. For these particular automata, $\epsilon$-rules can be eliminated in linear time, since no two $\epsilon$ rules can be composed.

It now remains to show that $q_A = q_{\mathtt{e}(A)}$ for every pNSTT $A$. We omit these details. □

## C. Expressiveness of NSTTs and pNSTTs

We show that deterministic NSTTs as well as deterministic pNSTTs capture the class of monadic MSO-definable queries.

### C.1. MSO-definable queries

In a logical perspective, a tree $t$ is seen as a logical structure with domain $\mathtt{nodes}(t)$, unary relations $\mathtt{label}_a$ for every $a \in \Sigma$, and two binary relations $\mathtt{firstchild}$ and $\mathtt{nextsibling}$. We assume an infinite set of first-order variables $x, y, z$, and an infinite set of monadic second-order variables $S$. Formulas of MSO in unranked (or binary) trees have the following syntax:

$$\phi ::= \mathtt{label}_a(x) \mid \mathtt{firstchild}(x, y) \mid \mathtt{nextsibling}(x, y)$$
$$\mid \quad \phi \to \phi' \mid S(x) \mid \forall x.\phi \mid \forall S.\phi$$

Standard logical connectives such as conjunction $\phi \wedge \phi'$, negation $\neg\phi$, existential quantification $\exists x.\phi$ and $\exists S.\phi$ can be expressed as usual, as well set inclusion $S \subseteq S'$. Further relations such as $\mathtt{child}$ and $\mathtt{lastchild}$ are definable in MSO as well.

Models of MSO formulas are defined in the usual Tarskian style. They consist of a tree structure $t$ and a variable assignment $\alpha$ mapping first-order variables to $\mathtt{nodes}(t)$ and second-order variables to $2^{\mathtt{nodes}(t)}$. We write $t, \alpha \models \phi$ if $t, \alpha$ is a model of $\phi$, and consider models of $\phi$ as equal if they coincide on the free variables of $\phi$.

Every MSO formula $\phi(x)$ with a single free variable $x$ defines a monadic query $q_{\phi(x)}$ in unranked (resp. binary) trees, so that for all trees $t$:

$$q_{\phi(x)}(t) = \{\alpha(x) \in \texttt{nodes}(t) \mid t, \alpha \models \phi(x)\}$$

C.2. NSTT and PNSTT-defined queries

The idea behind completely annotated examples (as recognized by NSTTs) is to annotate all nodes that are selected in some tree in a single annotation. In MSO, we can express the same idea by collecting all nodes $x$ selected by some query $\phi(x)$ in a single set $S$.

$$\phi'(S) \;\equiv\; \forall x.\, (S(x) \leftrightarrow \phi(x))$$

More generally, we can use MSO formulas $\phi(S)$ with a single free set variable $S$ to define monadic queries, such that for all trees $t$:

$$q_{\phi(S)}(t) = \{v \in \texttt{nodes}(t) \mid t, \alpha \models \phi(S),\; v \in \alpha(S)\}$$

A monadic query is definable by a MSO formula with a single free first-order variable if and only if it is definable by an MSO formula with a single free second-order variable.

Annotated trees can be understood as the models of MSO formulas with a single set valued variable. A model $t, \alpha$ of MSO formulas with a single free variable $S$ becomes the annotated tree $t \times \alpha(S)$, which annotates all nodes in $t$ belonging to $\alpha(S)$ by 1 and all others by 0.

**Theorem 4** (Thatcher & Wright, 1968). *The set of annotated trees corresponding to models of an MSO formula $\phi(S)$ over signature $\Sigma$ is a regular tree language over $\Sigma \times \mathsf{Bool}$, and vice versa.*

The theorem states that a monadic query in binary trees over $\Sigma$ definable by an MSO formula with one free set variable if and only it is definable by some tree automaton over the signature $\Sigma \times \mathsf{Bool}$. The original theorem is even more general in that it applies to $n$-ary queries which are definable by formulas with $n$ free variables (Niehren et al., 2005).

**Theorem 5.** *A query is definable in MSO iff it is definable by a deterministic* NSTT *iff it is definable by a deterministic* PNSTT.

**Proof:** Consider a query that is defined by some MSO-formula $\phi(x)$. The same query is defined by the formula $\phi'(S) \equiv \forall x(x \in S \leftrightarrow \phi(x))$. Formula $\phi'(S)$ is functional in that the value of $S$ is uniquely determined by the tree structure of models of $\phi'(S)$. Let $A$ be the tree automaton over $\Sigma \times \mathsf{Bool}$ that recognizes the set of annotated trees corresponding to the models of $\phi'(S)$, according to Thatcher and Wright's Theorem 4. The language of this automaton is functional, given that the value of $S$ in models $t, \alpha \models \phi'(S)$ is uniquely defined. This proves that $A$ is an NSTT. By construction, it defines the same query as $\phi'(S)$ and thus $\phi(x)$. The same automaton is also a PNSTT. □

We have shown in Section B.2 that monadic queries represented by deterministic NSTTs can be identified with run-based queries defined by unambiguous tree automata. Theorem 5 shows that runs of unambiguous tree automata can express all MSO definable monadic

queries. This result has been shown before by Neven and Van Den Bussche (2002). The simple MSO-based proof presented here stems from Niehren et al. (2005) where the expressiveness of $n$-ary queries by unambiguous tree automata has been studied too.

**Acknowledgments** We are grateful to Alain Terlutte for contributing his forces and enthusiasms into this work in the early phase.

We thank the referees for their invaluable comments on preliminary versions of the paper.

This research was partially supported by the research unit INRIA Futurs, CNRS UMR 8022, "CPER 2000-2006, Contrat de Plan état—région Nord/Pas-de-Calais: programme TAC, projet COCOA" and "ACI masse de données ACI-MDD, FNS".

## References

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, *75*(2), 87–106.

Baumgartner, R., Flesca, S., & Gottlob, G. (2001). Visual web information extraction with lixto. In *28th International Conference on Very Large Data Bases*, pp. 119–128.

Brüggemann-Klein, A., Wood, D., & Murata, M. (2001). Regular tree and regular hedge languages over unranked alphabets: Version 1..

Carme, J., Gilleron, R., Lemay, A., & Niehren, J. (2005). Interactive learning of node selecting tree transducer. In *IJCAI Workshop on Grammatical Inference*.

Carme, J., Lemay, A., & Niehren, J. (2004a). Learning node selecting tree transducer from completely annotated examples. In *7th International Colloquium on Grammatical Inference*, Vol. 3264 of *Lecture Notes in Artificial Intelligence*, (pp. 91–102). Springer Verlag.

Carme, J., Niehren, J., & Tommasi, M. (2004b). Querying unranked trees with stepwise tree automata. In *19th International Conference on Rewriting Techniques and Applications*, Vol. 3091 of *Lecture Notes in Computer Science*, (pp. 105–118). Springer Verlag.

Chidlovskii, B. (2001). Wrapping web information providers by transducer induction. In *Proc. European Conference on Machine Learning*, Vol. 2167 of *Lecture Notes in Artificial Intelligence*, pp. 61–73.

Cohen, W., Hurst, M., & Jensen, L. (2003). *Web document analysis: challenges and opportunities*, chap. A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. World Scientific.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., & Tommasi, M. (1997). Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata.

Cristau, J., Löding, C., & Thomas, W. (2005). Deterministic automata on unranked trees. In *15th International Symposium on Fundamentals of Computation Theory*. To Appear.

de la Higuera, C. (1997). Characteristic sets for polynomial grammatical inference. *Machine Learning*, *27*, 125–137.

Drewes, F., & Hogberg, J. (2003). Learning a regular tree language from a teacher. In *D.L.T. 2003*, Vol. 2710 of *Lecture Notes in Computer Science*, (pp. 279–291).

Freitag, D., & Kushmerick, N. (2000). Boosted wrapper induction. In *AAAI/IAAI*, (pp. 577–583).

Freitag, D., & McCallum, A. K. (1999). Information extraction with hmms and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*.

Frick, M., Grohe, M., & Koch, C. (2003). Query evaluation on compressed trees. In *18th IEEE Symposium on Logic in Computer Science*, (pp. 188–197).

Gold, E. (1967). Language identification in the limit. *Inform. Control*, *10*, 447–474.

Gold, E. (1978). Complexity of automaton identification from given data. *Inform. Control*, *37*, 302–320.

Gottlob, G., & Koch, C. (2002). Monadic queries over tree-structured data. In *17th Annual IEEE Symposium on Logic in Computer Science*, (pp. 189–202) Copenhagen.

Hsu, C.-N., & Dung, M.-T. (1998). Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, *23*(8), 521–538.

Kermorvant, C., & de la Higuera, C. (2002). Learning language with help. In *6th International Colloquium on Grammatical Inference*, Vol. 2484 of *Lecture Notes in Artificial Intelligence*, (pp. 161–173). Springer Verlag.

Kosala, R., Bruynooghe, M., Van den Bussche, J., & Blockeel, H. (2003). Information extraction from web documents based on local unranked tree automaton inference. In *18th International Joint Conference on Artificial Intelligence*, (pp. 403–408). Morgan Kaufmann.

Kushmerick, N. (1997). *Wrapper Induction for Information Extraction*. Ph.D. thesis, University of Washington.

Kushmerick, N. (2000). Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, *118*(1–2), 15–68.

Kushmerick, N. (2002). Finite-state approaches to web information extraction. In *Proc. 3rd Summer Convention on Information Extraction*.

Lang, K. J., Pearlmutter, B. A., & Price, R. A. (1998). Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science*, *1433*, 1–12.

Lang, K. (1992). Random DFA's can be approximately learned from sparse uniform examples. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, (pp. 45–52). ACM Press, New York, NY.

Libkin, L. (2005). Logics over unranked trees: An overview. In *Automata, Languages and Programming: 32nd International Colloquium*, No. 3580 in Lecture Notes in Computer Science, (pp. 35–50). Springer Verlag.

Martens, W., & Niehren, J. (2006). On the minimization of XML schemas and tree automata for unranked trees. *Journal of Computer and System Science*. Special issue of DBPL'05.

Muslea, I., Minton, S., & Knoblock, C. A. (2001). Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, *4*(1/2), 93–114.

Neumann, A., & Seidl, H. (1998). Locating matches of tree patterns in forests. In *Foundations of Software Technology and Theoretical Computer Science*, (pp. 134–145).

Neven, F., & Schwentick, T. (2002). Query automata over finite trees. *Theoretical Computer Science*, *275*(1–2), 633–674.

Neven, F., & Van Den Bussche, J. (2002). Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, *49*(1), 56–100.

Niehren, J., Planque, L., Talbot, J.-M., & Tison, S. (2005). N-ary queries by tree automata. In *10th International Symposium on Database Programming Languages*, Vol. 3774 of *Lecture Notes in Computer Science*, (pp. 217–231). Springer Verlag.

Oncina, J., & García, P. (1992). Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, (pp. 49–61).

Oncina, J., & García, P. (1993). Inference of recognizable tree sets. Tech. rep., Departamento de Sistemas Informáticos y Computación, Universidad de Alicante. DSIC-II/47/93.

Raeymaekers, S., & Bruynooghe, M. (2004). Minimization of finite unranked tree automata. Manuscript.

Raeymaekers, S., Bruynooghe, M., & Van den Bussche, J. (2005). Learning (*k,l*)-contextual tree languages for information extraction. In *Proceedings of ECML'2005*, Vol. 3720 of *Lecture Notes in Artificial Intelligence*, (pp. 305–316).

Sakakibara, Y. (1990). Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, *76*, 223–242.

Seidl, H. (1989). On the finite degree of ambiguity of finite tree automata. *Acta Informatica*, *26*(6), 527–542.

Thatcher, J. W. (1967). Characterizing derivation trees of context-free grammars through a generalization of automata theory. *Journal of Computer and System Science*, *1*, 317–322.

Thatcher, J. W., & Wright, J. B. (1968). Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, *2*, 57–82.