Query Induction with Schema-Guided Pruning Strategies

Joachim Niehren, Jérôme Champavère, Rémi Gilleron, Aurélien Lemay

Query Induction with Schema-Guided Pruning Strategies

Joachim Niehren joachim.niehren@inria.fr
Jérôme Champavère jerome.champavere@lifl.fr
Rémi Gilleron remi.gilleron@univ-lille3.fr
Aurélien Lemay aurelien.lemay@univ-lille3.fr
Mostrare – Inria *Lille &* Lifl – *France*

Editor: Mehryar Mohri

Abstract

Inference algorithms for tree automata that define node selecting queries in unranked trees rely on tree pruning strategies. These impose additional assumptions on node selection that are needed to compensate for small numbers of annotated examples. Pruning-based heuristics in query learning algorithms for Web information extraction often boost the learning quality and speed up the learning process. We will distinguish the class of regular queries that are stable under a given schema-guided pruning strategy, and show that this class is learnable with polynomial time and data. Our learning algorithm is obtained by adding pruning heuristics to the traditional learning algorithm for tree automata from positive and negative examples. While justified by a formal learning model, our learning algorithm for stable queries also performs very well in practice of xml information extraction.

Keywords: xml information extraction, xml schemas, interactive learning, tree automata, grammatical inference

1. Introduction

A fundamental problem of xml information extraction is query induction from annotated examples. The problem is to select "relevant" elements in collections of xml documents, while knowing only few positive and negative examples for relevant elements. The target of this learning problem is thus a query for elements in xml documents.

Most approaches for xml information extraction can be found in the context of Web information extraction. The many learning methods applied there range from statistical classification (Kushmerick, 2000; Gilleron et al., 2006b), hidden Markov models (Freitag and McCallum, 1999), conditional random fields (Pinto et al., 2003; Zhu et al., 2005; Gilleron et al., 2006a), active learning on strings (Muslea et al., 2003), grammatical inference (Raeymaekers et al., 2008; Carme et al., 2007), to inductive logic programming (Cohen et al., 2002). Queries for information extraction can also be produced by visual programming (Baumgartner et al., 2001) possibly with learning enhancement (Carme et al., 2006a). Unsupervised approaches based on ontology knowledge were proposed recently (Sellers et al., 2011a).

For illustration, we consider in Figure 1 an xml document for a geographical database with information about regions in France. This xml document can be parsed into the data tree in Figure 2. A user may then want to select, for instance, all those regions for which the size of the population is known in the database. This goal can be translated to a query to

```
<country>
  <name>France</name>
  <city>Paris</city>
  <region>
    <name>Nord-Pas de Calais</name>
    <population>3 996 588</population>
    <city>Lille</city>
  </region>
  <region>
    <name>Vallée du Rhône</name>
    <city>Lyon</city>
    <city>Valence</city>
  </region>
</country>
```

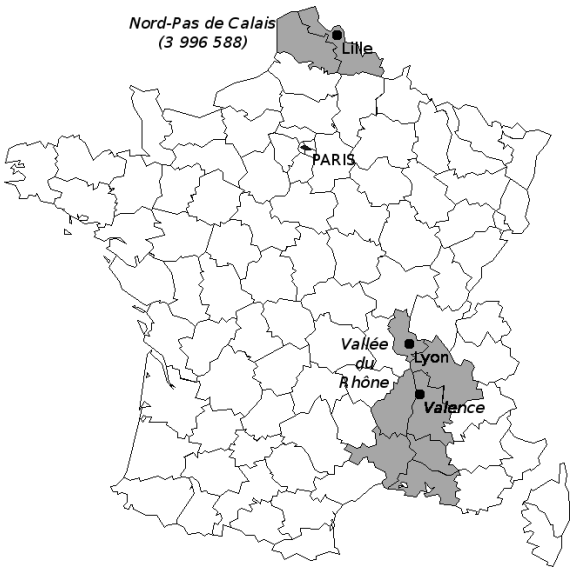


Figure 1: A tiny geographical XML database.

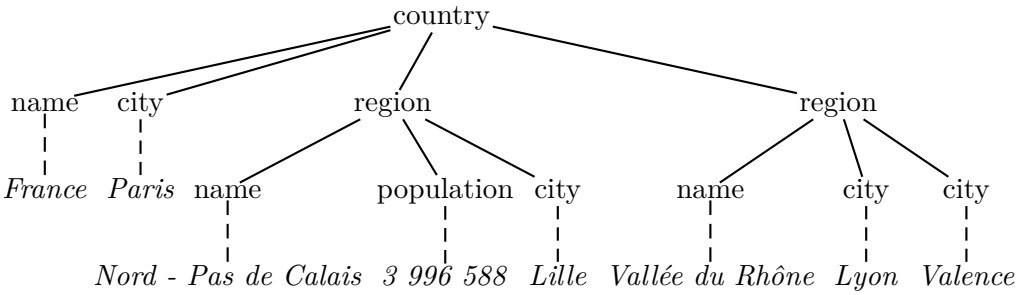


Figure 2: Data tree of the geographical database in Figure 1.

data trees of geographical databases, which selects all nodes labeled by `region` and having a child labeled by `population`, that is the XPath query `//region[population]`.

However, finding this query requires knowledge on XPath and the XML schema of geographical databases, which is not to be expected from a non-expert user. The difficulties of non-expert users can be solved by supervised query induction. The idea is that the user annotates some examples of elements positively or negatively, meaning that they should be selected or not. This can be done in a graphical interface of the database, as illustrated in Figure 3, or in a Web browser, and then translated to annotations on the XML tree. It can also be done directly by annotating the XML tree.

The query induction algorithm then learns the query from the XML trees with some annotated elements. The main difficulties of query induction are the following:

1. the availability of only few annotated examples,

2. missing semantic information about the meaning of XML tags, and

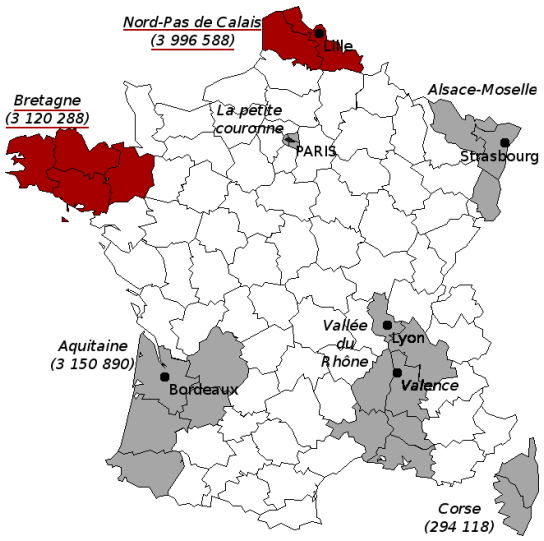


Figure 3: Two positive examples annotated via the user interface.

3. the difficulty to understand natural language texts in data trees.

We do not tackle the third problem in the present paper. Instead, we restrict ourselves to node selection queries in unranked trees without data values. In order to limit the burden of annotation and to obtain relevant annotations quickly, we will rely on interactive learning in Angluin style (Angluin, 1987), where selected nodes proposed by the learner are corrected by the teacher in an annotate-learn-select loop. This corresponds to the usual user interaction loop of Web information extraction systems (see, e.g., Muslea et al. (2003)).

In this article, we study the relevance of schema-guided pruning heuristics for query induction[1]. Pruning consists in removing useless information for learning. Schema-less pruning strategies were essential for good quality with few examples of query induction algorithms based on tree automata inference by, e.g., Raeymaekers et al. (2008) or Carme et al. (2007), and also for decent efficiency. So far, however, pruning strategies were always defined in ad hoc manners. Thus, our first objective is to define them systematically. Pruning strategies must be aggressive on the one hand side, since the more subtrees are pruned, the more efficient the learning algorithm will be. On the other hand side, pruning strategies must preserve the information required to learn the query. Our second objective is hence to formalize the relationship between pruning strategies and learnable classes of queries. Finally, schemas express semantic information on XML trees which should allow to learn larger classes. Our third objective is thus to use schemas to improve learning algorithms and to understand the relevance of schemas for pruning strategies in particular.

Document type definitions (DTDs) are the simplest schemas for XML documents. For instance, let us consider the DTD in Figure 4 for our geographical database example. It says that under the root with label country, there is its name, followed by a city, followed by a possibly empty list of regions. Each region is described by its name,

1. Early ideas of the present article were published by Champavère et al. (2008) at ICGI, elaborated in the PhD thesis of Champavère (2010), and seriously revised here.

$$
\begin{array}{lll}
\mathit{country} & \rightarrow & \mathit{name} \cdot \mathit{city} \cdot \mathit{region}^* \\
\mathit{region} & \rightarrow & \mathit{name} \cdot \mathit{population}^\epsilon \cdot \mathit{city}^+ \\
\mathit{name} & \rightarrow & \epsilon \\
\mathit{city} & \rightarrow & \epsilon \\
\mathit{population} & \rightarrow & \epsilon
\end{array}
$$

Figure 4: Schema *Geo* for a geographical XML database restricted to the description of a country. It is defined by a DTD where country is the label of the root.

its population but it is not mandatory, and a non empty list of cities. For sake of simplicity, we assume that subtrees with root label name, city and population only contain data values that we do not consider here. For instance, the unranked tree $u = \text{country}(\text{name}, \text{city}, \text{region}(\text{name}, \text{population}, \text{city}), \text{region}(\text{name}, \text{city}, \text{city}))$ is valid for the DTD *Geo*, it is the tree presented in Figure 2 without the data values. Schemas can also be defined by deterministic tree automata. These are finite state machines adapted to trees. They process trees in a bottom-up way (from the leaves to the root) according to a finite set of rules. The result is a tree annotated with states. For instance, a tree automaton associated with the DTD *Geo* can be defined with states $q_{country}$, q_{name}, etc., and a state $q_{invalid}$, and rules expressing the conditions given by the DTD. For an invalid tree, the state $q_{invalid}$ will appear along the run of the automaton and the tree will be rejected, while a valid tree will be processed and annotated by the correct states. For instance the unranked tree u will be annotated as $q_{country}(q_{name}, q_{city}, q_{region}(q_{name}, q_{population}, q_{city}), q_{region}(q_{name}, q_{city}, q_{city}))$, and will be accepted because the root state is $q_{country}$.

We next introduce schema-guided pruning strategies by example and illustrate their relationship to query classes. Pruning strategy *path-only* applied to a node of the XML database keeps only the path leading from the root to the node, and replaces all subtrees adjacent to this path by a special symbol $\top$. Pruning strategy $\textit{path-only}_{Geo}$ keeps that same path but replaces the adjacent subtrees by the state assigned to them by the tree automaton for *Geo*. In Figure 5, the result of applying both pruning strategies to the `name`-child of the `region`-node in Figure 2 is presented. It will turn out that more complex pruning strategies will be needed for ranked trees, in order to deal with the above pruning strategies for unranked trees via binary encoding.

We call a query *stable* for a pruning strategy if the result of applying the strategy to a tree at some selected node does always justify the node's selection. In this case, we call this result the critical region for selecting the node. For illustration, let us consider the query that selects all names of regions whose population is known in our geographic XML database with schema *Geo*, that is the XPath query `//region[population]/name`. Whether a node with label `name` is selected depends only on its local environment, more precisely, on whether its father is labeled by `region` and whether its right-sibling is labeled by `population`. It is easy to see that pruning strategy *path-only* applied to a selected node removes the label of its right sibling. Thus this query is not stable for pruning strategy *path-only*. In contrast, the schema-guided pruning strategy $\textit{path-only}_{Geo}$ stores the label of

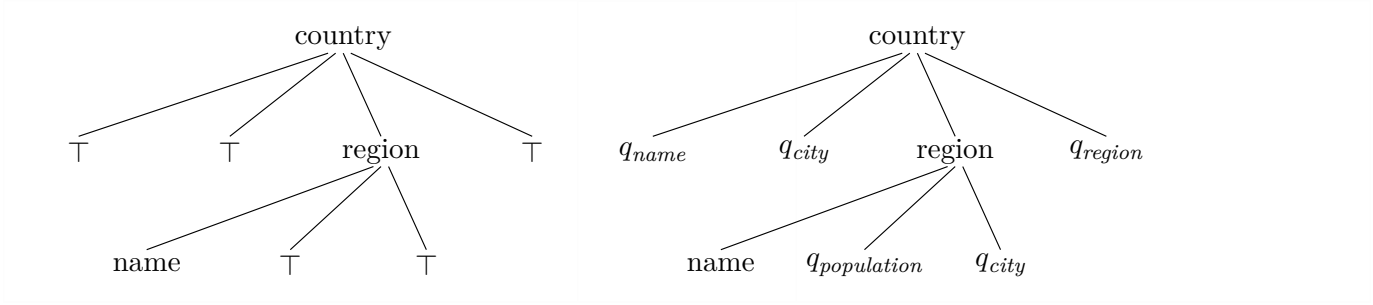


Figure 5: Pruning the tree from Figure 2 at the name-child of the first region-node, by strategies *path-only* (left) and $path\text{-}only_{Geo}$ (right).

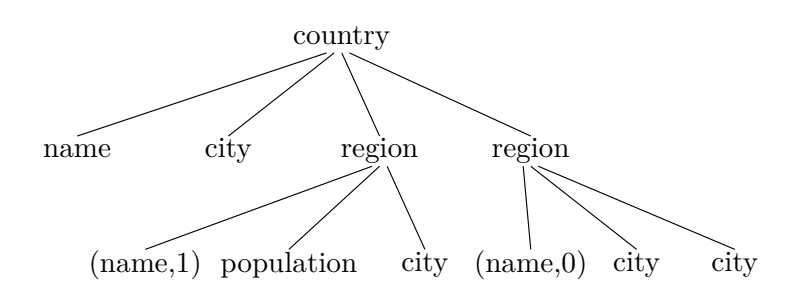


Figure 6: An annotated tree in which the first region name has been annotated positively (it must be selected) and the second region name has been annotated negatively (it must not be selected).

the right sibling in state $q_{population}$, so that this relevant information is preserved. Therefore, the query `//region[population]/name` is stable for pruning strategy $path\text{-}only_{Geo}$.

A user of the learning system may want to learn the above XPath query since he does not know how to express it formally. He might not be an XPath expert or might not have access to the schema *Geo* of our geographic database. Such users may still be willing to annotate some selected `name`-nodes by the Boolean value 1 and some rejected nodes by the Boolean value 0 with the help of a graphical interface. An example for a partial annotation of our geographical database that could be obtained this way is given in Figure 6. Pruning strategies can be lifted to pruning functions on positively annotated trees. The easiest way to do so is to keep only the union of the critical regions of all nodes annotated by 1 jointly with their annotations.

Given a pruning strategy σ, the next question is whether σ-stable queries can be learned from *σ-pruned samples* of annotated examples. These are finite sets that contain σ-pruned trees with positive annotations and unpruned trees with negative annotations, such that all annotations are consistent with respect to the target query. For any pruning strategy σ, we

will distinguish the class of regular σ-stable queries and show that this class can indeed be identified from σ-pruned samples. Depending on the pruning strategy, this yields a hierarchy of query classes that is essential for understanding the difficulty of query learning in practice.

Contributions. We study the impact of pruning strategies on learning algorithms for classes of regular queries for the first time. We recall that XPath queries (without tests on data values) are regular, even if imposing schema restrictions by DTDs or tree automata.

1. We define schema-guided pruning strategies for schemas defined by deterministic bottom-up tree automata, both for ranked and unranked trees. We introduce the notion of stable queries for a given pruning strategy. We show how to characterize stable queries by languages of pruned annotated trees. For regular queries these languages can be recognized by tree automata.

2. We lift pruning strategies to pruning functions that can be applied to positively annotated trees, in order to produce pruned samples for our learning algorithm.

3. We show how to represent pruning functions by means of tree automata. Thereby we define the notion of regular pruning functions. We present an algorithm that decides in polynomial time whether a regular query is stable for a regular pruning function.

4. We present a learning algorithm, based on the $\mathcal{RPNI}$ algorithm (García and Oncina, 1993), that identifies regular σ-stable queries in polynomial time from σ-pruned samples of polynomial cardinality.

5. We present experimental results that confirm the relevance of our query learning algorithm in an interactive learning environment, both for Web information extraction and for XML information extraction.

6. We compare different pruning strategies with respect to their aggressiveness. It turns out that more aggressive pruning strategies yield better learning quality. We also discuss how to select appropriate schema-guided pruning strategies in practice.

Outline. For sake of clarity, we first give definitions, results and proofs for ranked trees. All results can be lifted to the case of unranked trees via a binary encoding. Experiments will be done on information extraction tasks over unranked trees (HTML trees or XML trees).

Section 2 recalls preliminaries on tree automata for ranked trees and illustrates how to use them as schemas. Section 3 introduces the notion of stable queries for schema-guided pruning strategies and shows that less aggressive pruning strategies give rise to larger classes of stable queries. In Section 4 we show how to lift pruning strategies to pruning functions, by which to prune examples with positive annotations only. We then show how to characterize stable queries for pruning functions by regular languages of pruned annotated examples in a unique manner. Thereby we specify the target languages for the learning algorithm. Section 5 discusses how to define regular stable queries and pruning functions by deterministic tree automata. In Section 6 we present algorithms for testing two consistency properties for regular languages of pruned annotated trees. Section 7 presents our new learning algorithm for stable queries from pruned annotated examples based on these algorithms. In Section 8, we lift all previous results to unranked trees via a binary encoding. Experimental results on

query induction in XML information extraction are discussed in Section 9. Future work and conclusions are presented in Section 10. The appendix completes the proofs of results that are not essential for the main contributions.

Related Work. There exist two previous approaches for inducing regular languages that can account for the consistency with some domain of the target language. Both of them guarantee that the current hypothesis L of the learning algorithm does satisfy $L \subseteq \mathcal{L}(D)$, were D is a deterministic finite automaton defining the domain, in which the target language must be included. The first dynamic approach is to test language inclusion after each generalization step (Coste et al., 2004). This is the approach we here generalize to query induction. The second static approach is to ensure inclusion (Oncina and Varó, 1996) by typing all states of the current DFA A by the states of the domain DFA D. This means that the target automaton will be a product with the domain automaton D. Formal learnability results were missing for both approaches so far. In the present article, we provide such results for the first time for the dynamic approach. Furthermore, we show that the dynamic approach is feasible in practice even in the case of tree automata, where inclusion testing is more tedious. The static approach performs worse in our applications in most cases.

Schema induction for XML documents was invested by Bex et al. (2006). The interest there is to produce readable DTDs with regular expressions. Query induction, as presented here, usually tries to avoid the induction of schemas, since queries may only rely on parts of the schema. Conversely, however, one might use induced schemas for query induction. Induction of XPath queries was considered by Carme et al. (2006b) and Staworko and Wieczorek (2011). Induction algorithms for top-down deterministic tree transducers were presented by Lemay et al. (2010).

2. Schemas and Tree Automata

In this section, we recall facts on tree automata on ranked trees and illustrate how to use them as schemas. Unranked trees will be treated later on via a binary encoding. Most results we will remind here are standard with the exception of an efficient inclusion test, which will be fundamental for the efficiency of our schema-guided learning algorithm.

Let $\mathbb{N}$ be the set of non-zero natural numbers. A ranked signature is a set Σ equipped with a function rank_Σ from Σ to $\mathbb{N} \cup \{0\}$. For $k \geq 0$, we denote by $\Sigma^{(k)}$ the set $\{f \in \Sigma \mid \text{rank}_\Sigma(f) = k\}$. We write $f^{(k)}$ to indicate that f is of rank k. The set $\mathcal{T}_\Sigma$ of ground terms over Σ is the least set that contains all tuples $f(t_1, \ldots, t_k)$ where $f \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in \mathcal{T}_\Sigma$. Ground terms in $\mathcal{T}_\Sigma$ are equivalently called ranked Σ-trees or simply trees. As usual we write f instead of the single node tree $f()$ where $f^{(0)} \in \Sigma$ is a constant. We denote by $\mathit{nodes}(t) \subseteq \mathbb{N}^*$ the (prefix-closed) set of nodes of the tree t. The label of a node ν in t is denoted by $t[\nu]$. Given a subset $\Sigma' \subseteq \Sigma$ of labels, we denote by $\mathit{nodes}_{\Sigma'}(t) = \{\nu \in \mathit{nodes}(t) \mid t[\nu] \in \Sigma'\}$ the set of all nodes of t labeled in Σ'.

A *tree automaton* is a tuple $D = (\Sigma, X, R, F)$ where Σ is a finite ranked signature, X is a finite set of states, $F \subseteq X$ is a set of final states, and $R \subseteq \bigcup_{k \geq 0} \Sigma^{(k)} \times X^{k+1}$ is a set of (transition) rules. We denote by $|D|$ the sum of the lengths of the rules of D and call it the size of D. A transition rule $r \in R$ is a tuple $(f^{(k)}, q_1, \ldots, q_k, q)$ where $q, q_1, \ldots, q_k$ are states in X. As usual, we denote such a rule r by $f(q_1, \ldots, q_k) \to q$ and call $f(q_1, \ldots, q_k)$ its

left-hand side. A tree automaton D is (bottom-up) *deterministic* if no two rules of D have the same left-hand side.

The evaluator of D is a function $eval_D : \mathcal{T}_\Sigma \to 2^X$ such that for all trees $f(t_1, \ldots, t_k) \in \mathcal{T}_\Sigma$: $eval_D(f(t_1, \ldots, t_k)) = \{q \mid f(q_1, \ldots, q_k) \to q \text{ in } D, \forall i \in \{1, \ldots, k\} : \ q_i \in eval_D(t_i)\}$. If D is deterministic, then, for any t, the set $eval_D(t)$ contains at most one element. The language recognized by D is the set of all trees in $\mathcal{T}_\Sigma$ that D can evaluate into some final state:

$$\mathcal{L}(D) = \{t \in \mathcal{T}_\Sigma \mid eval_D(t) \cap F \neq \emptyset\}.$$

A tree language $L \subseteq \mathcal{T}_\Sigma$ is *regular* if $L = \mathcal{L}(D)$ for some tree automaton D with signature Σ.

Example 1 *We consider libraries (lib) which contain lists of books (b) with lists of authors (a). We use cons and nil as list constructors as usual for ranked trees, so we use the following signature* $\Sigma_{Lib} = \{lib^{(1)}, b^{(1)}, a^{(0)}, cons^{(2)}, nil^{(0)}\}$. *The unranked library* $lib(b(a, a), (b(a, a), b))$ *with three books of which the last one has no author is represented by the following ranked* Σ*-tree, which is also illustrated graphically on the left of Figure 8:*

$$lib(cons(b(cons(a, cons(a, nil))), cons(b(cons(a, cons(a, nil))), cons(b(nil), nil))))$$

The schema of libraries is defined by the tree automaton Lib' *with signature* Σ_{Lib} *with state set* $X = \{q_{lib}, q_{bs}, q_{as}, q_b, q_a\}$*, final state* q_{lib}*, and the rules:*

$$\begin{array}{llllllll} lib(q_{bs}) & \to & q_{lib} & b(q_{as}) & \to & q_b & nil & \to & q_{bs} \\ cons(q_b, q_{bs}) & \to & q_{bs} & nil & \to & q_{as} & cons(q_a, q_{as}) & \to & q_{as} \\ a & \to & q_a & & & & & & \end{array}$$

Note that Lib' *is nondeterministic, since symbol nil may be analyzed as the end of a book-list or as the end of an author-list. Let Lib be the determinization of* Lib'*. This can be done by mapping nil to some new state* $\{q_{as}, q_{bs}\}$*, while adding the necessary rules for this state.*

We will need some basic constructions on tree automata and complexity results which can be found in (Comon et al., 2007). Given a tree automaton D, it is decidable in time $O(|D|)$ whether $\mathcal{L}(D) = \emptyset$. Given another tree automaton D' over the same signature, a tree automaton $D \cap D'$ with $\mathcal{L}(D \cap D') = \mathcal{L}(D) \cap \mathcal{L}(D')$ can be computed in time $O(|D| \ |D'|)$.

Our learning algorithm will heavily rely on language inclusion tests for deterministic tree automata. We will use a recent inclusion test, proposed by the authors, which can be implemented highly efficiently and incrementally (w.r.t. adding rules to D'):

Proposition 1 (Efficient Inclusion Test (Champavère et al., 2009)) *Let* D' *and* D *be tree automata over* Σ*. If* D *is deterministic then language inclusion* $\mathcal{L}(D') \subseteq \mathcal{L}(D)$ *can be decided in time* $O(|D'| \ |D|)$*.*

3. Schema-Guided Pruning Strategies

We introduce schema-guided pruning strategies and define a partial aggressiveness order on them. We then present stable queries for a given pruning strategy. We show that less aggressive pruning strategies give rise to larger classes of stable queries.

3.1 Pruned Trees

We fix a ranked signature Σ and a schema D which is a deterministic tree automaton with signature Σ and state set X. We define a *pruned tree* to be a tree over signature $\Sigma \cup X$, i.e., the states of D become additional constants. Whenever D is unclear from the context, we will talk of *D-pruned trees* equivalently to pruned trees.

We will write $\mathcal{T}_\Sigma(X)$ instead of $\mathcal{T}_{\Sigma \cup X}$ for the set of pruned trees. It should be noted that schema D can be also abused to recognize pruned trees. In order to do so, it is sufficient to add the rules $q \to q$ for all states $q \in X$. In this way, the evaluator $eval_D$ for unpruned trees can be lifted to pruned trees.

Let t and t' be two pruned trees in $\mathcal{T}_\Sigma(X)$. We say that t *is subsumed by* t', or equivalently that t' *is a D-instance of* t, and write $t \leq t'$ if t' can be obtained from t by replacing occurrences of states q by pruned trees that can be evaluated to q by D. We call a finite set of pruned trees $L \subseteq \mathcal{T}_\Sigma(X)$ *compatible* (with respect to D) if all trees in L have a common instance, i.e., if there exists a tree t' such that $t \leq t'$ for all $t \in L$. In this case, L has a least upper bound, that we denote by $\sqcup L$ such that $t \leq \sqcup L$ for all $t \in L$. The existence of least upper bounds follows, since we assume that D is deterministic, so that no subsets of X with two different states $\{q, q'\} \subseteq X$ are compatible. For instance, if $\Sigma = \{f^{(2)}, g^{(1)}, a^{(0)}\}$ and D contains the rules $a \to q'$, $g(q') \to q$, and $f(q, q) \to q''$ where q'' is final then:

$$\sqcup\{f(g(a), q), f(q, g(q')), f(q, q), q''\} = f(g(a), g(q'))$$

A D-completion of a pruned tree $t \in \mathcal{T}_\Sigma(X)$ is a tree $t' \in \mathcal{T}_\Sigma$ such that $t \leq t'$. Every pruned tree t defines a sublanguage of $\mathcal{T}_\Sigma$ that we denote by $compl_D(t)$ containing all its D-completions. Note that $compl_D(q)$ is the set of all trees $t \in \mathcal{T}_\Sigma$ that D can evaluate to q.

3.2 Pruning Strategies

The purpose of a pruning strategy is to remove all parts of a tree that are irrelevant for the selection of a given node. Removed subtrees are always replaced by the state into which they are evaluated by the schema, so that some information about removed subtrees may still be preserved.

Definition 2 *A pruning strategy for D is a function σ that maps any tree $t \in \mathcal{L}(D)$ and node $\nu \in nodes(t)$ to a pruned tree $\sigma(t, \nu) \in \mathcal{T}_\Sigma(X)$ of which t is a D-instance, such that ν is preserved with its label.*

Since schema D is a deterministic tree automaton, there is no choice by which state a subtree is to be replaced. Thus, a pruning strategy can be specified by the subset of nodes rooting pruned subtrees. Let us introduce the four pruning strategies that will be used in the experiments. The pruning strategy *path-only* is total. It prunes away all maximal subtrees that are not ancestors of ν and replaces them by $\top$. The pruning strategy *path-only*$_D$ is more restrictive in that it can only be applied to trees satisfying schema D. Note that if $\mathcal{U}$ is the unique universal tree automaton with a single state $\top$ that recognizes all trees, then *path-only* = *path-only*$_\mathcal{U}$. Pruning strategy *path-ext*$_D$ can be applied to any tree satisfying schema D. When applied to a tree t with node ν, the extended path from the root to ν remains unchanged, i.e., all children of siblings of nodes on the path to ν are substituted by

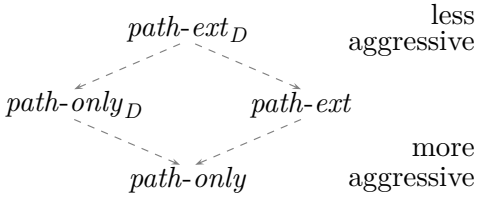


Figure 7: Example pruning strategies ordered by aggressiveness.

their state, as well as all children of ν. The pruning strategy *path-ext* is equal to *path-ext*$_D$ where $D = \mathcal{U}$.

Definition 3 *Let σ and σ' be two pruning strategies for schema D and D' respectively such that $\mathcal{L}(D) \subseteq \mathcal{L}(D')$. We call σ less aggressive than σ' if any tree $t \in \mathcal{L}(D)$ with node ν satisfies that $compl_D(\sigma(t,\nu)) \subseteq compl_{D'}(\sigma'(t,\nu))$.*

We will use the following sufficient criterion to prove that σ is less aggressive than σ':

1. Schema D should always evaluate to more informative states than D', i.e., for any tree t and state $q \in eval_D(t)$, there exists a state $q' \in eval_{D'}(t)$ such that $compl_D(q) \subseteq compl_{D'}(q')$, and
2. strategy σ should always replace fewer subtrees than σ', i.e., for any $t \in \mathcal{L}(D)$ with node ν: $nodes_\Sigma(\sigma(t,\nu)) \supseteq nodes_\Sigma(\sigma'(t,\nu))$.

Clearly, *path-ext*$_D$ is less aggressive than *path-only*$_D$ since *path-only*$_D$ always leaves the minimal number of nodes unchanged, but *path-ext*$_D$ leaves children of ancestor nodes on the path unchanged too. The strategy *path-only* is more aggressive than *path-only*$_D$ since both prune the same nodes away, while *path-only*$_D$ substitutes them by more informative states than *path-only*.

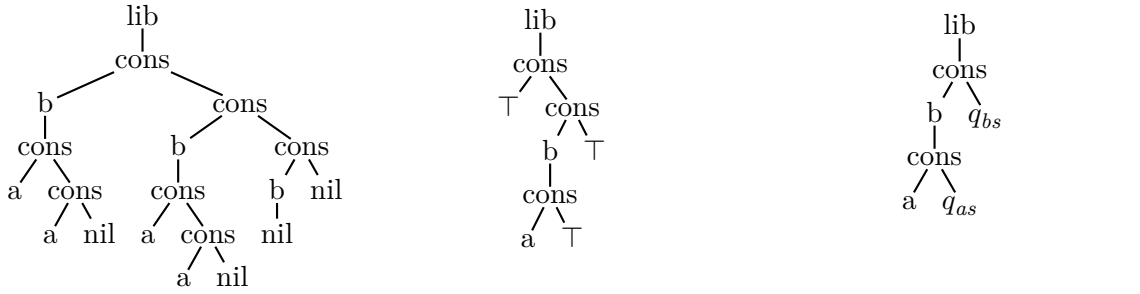
The notion of aggressiveness defines a partial order on pruning strategies, which is illustrated for our example strategies in Figure 7. This order is usually not total since *path-only*$_D$ and *path-ext* are incomparable for most choices of D and Σ.

3.3 Stable Queries

A *query Q with schema D* is a partial function with domain $\mathcal{L}(D)$ which maps trees t from this domain to sets of nodes $Q(t) \subseteq nodes(t)$. Note that queries can be applied to complete trees only. Now, we define the class of those queries that are stable with respect to a given schema-guided pruning strategy.

Definition 4 *Let σ be a pruning strategy and Q a query, both with schema D. We call Q σ-stable if all trees $t \in \mathcal{L}(D)$, selected nodes $\nu \in Q(t)$, and D-completions t' of $\sigma(t,\nu)$ satisfy $\nu \in Q(t')$.*

For a selected node $\nu \in Q(t)$ of a σ-stable query Q, we call $\sigma(t,\nu)$ the critical region of ν in t. Note that we do not define any critical region for rejected nodes, since the definition of stability talks only about selected nodes. The next example illustrates the relevance of selected nodes' critical regions.



an unpruned tree

path-only pruning for first author of the second book

$path\text{-}only_{\mathrm{Lib}}$ pruning for first author of the first book

Figure 8: Illustration of "path-only" pruning strategies at the library from Example 1 without schema and with schema Lib.

Example 2 *Let Lib be the deterministic tree automaton recognizing libraries from Example 1. On the left of Figure 8, we present an example library (lib), which contains three books (b), of which the first two have two authors (a), while the last has none. Given that we use ranked trees, lists are build by a binary list constructor (cons), and a constant (nil) for the empty list. We consider the following queries on libraries with this schema:*

Q_1 *selects the first author of all books of the library.*

Q_2 *selects the first author of all books of the library with at least two authors.*

Query Q_1 is stable for both pruning strategies path-only and thus for $path\text{-}only_{Lib}$ while query Q_2 is stable only for $path\text{-}only_{Lib}$ but not for path-only. This becomes clear when considering the applications of these pruning strategies to the nodes, that correspond to the first authors of the first two books.

The result of applying path-only to the first author of the second book is illustrated in the middle of Figure 8. Note that all the information needed for the selection of this author by Q_1 is preserved. However, since the second author of the same book is pruned away and replaced by $\top$, one cannot know whether there was a second author before, so necessary information is lost for deciding selection by Q_2.

The result of applying $path\text{-}only_D$ to the first author of the first book is shown on the right of Figure 8. This time, the pruned tree contains enough information to select the first author of the first book by Q_2, since now, the second author of this book is replaced by the state q_{as} which stands for a nonempty author list.

Our next objective is to restrict pruning strategies to schemas recognizing smaller languages. So let σ' be a pruning strategy with some schema D' such that $\mathcal{L}(D) \subseteq \mathcal{L}(D')$. We now define the pruning strategy $\sigma'_{|D}$ with schema D such that it replaces for all trees $t \in \mathcal{L}(D)$ the same subtrees t_0 as σ' by the unique state in $eval_D(t_0)$, but not by the unique state in $eval_{D'}(t_0)$ as chosen by σ'. Note that we deliberately overload the notation of function restriction here, in that states in images are changed too when restricting domains of pruning strategies.

Proposition 5 *Let σ and σ' be pruning strategies with respective schemas D and D' such that σ is less aggressive than σ'. Any query Q with schema D then satisfies that:*

$$Q \text{ is } \sigma'_{|D}\text{-stable} \Rightarrow Q \text{ is } \sigma\text{-stable.}$$

Proof Let tree t_0 be a D-completion of $\sigma(t, \nu)$ for some selected node $\nu \in Q(t)$. Since σ is less aggressive then σ', t_0 is also a D'-completion of $\sigma'(t, \nu)$. Since $t_0 \in L(D)$ and $\mathcal{L}(D) \subseteq \mathcal{L}(D')$, it is also a D-completion of $\sigma'_{|D}(t, \nu)$. The $\sigma'_{|D}$-stability of Q thus yields $\nu \in Q(t_0)$, so that ν remains selected on any σ-variant of t at ν, which shows that Q is also σ-stable. ■

4. Languages of Annotated Examples for Stable Queries

We will characterize stable queries in terms of languages of pruned positively annotated trees. In order to do so, we will lift pruning strategies to pruning functions that can be applied to example trees with positive annotations. There are several manners to do so, depending of whether only a single positive annotation is permitted or else many of them. Only once the choice of the pruning function is fixed, the characteristic language of a stable query can be defined in a unique manner.

The main idea of the learning algorithm for regular stable queries in Section 7 will be to identify the characteristic languages of the target query from annotated examples. Different methods for lifting pruning strategies to pruning functions will give rise to different target languages and thus to different learning algorithms.

4.1 Annotated Trees

Intuitively, annotated examples for a target query are trees in which some selected nodes are annotated by the Boolean 1 ("true") and some rejected nodes by the Boolean 0 ("false"). We will support partial annotations, so that only few annotations have to be added by a user. Nodes without annotation may either be selected or rejected. Some subtrees may be pruned away and replaced by a state of the schema of the query.

We next formalize the notion of annotated trees (independently of any target query or pruning strategy). Let $\mathbb{B} = \{0, 1\}$ be the set of Booleans. As before, we fix a ranked signature Σ and a deterministic tree automaton D with state set X. An *annotated tree* is a tree with ranked signature $\Sigma \cup (\Sigma \times \mathbb{B}) \cup X$, where all $q \in X$ have arity 0 while Boolean annotations preserve the arity. Nodes labeled by states in X are placeholders for subtrees which may contain both selected or rejected nodes. For instance, $(f, 0)(a, f((b, 1), q))$ is an annotated tree where $f^{(2)}, a^{(0)}, b^{(0)} \in \Sigma$ and $q \in X$. We call an annotated tree *unpruned* if none of its nodes is labeled in X. We call it *positively annotated* if none of its nodes is labeled in $\Sigma \times \{0\}$ and *negatively annotated* if none its nodes is labeled in $\Sigma \times \{1\}$.

An *annotation of a pruned tree* $t \in \mathcal{T}_\Sigma(X)$ is a partial function β mapping a subset of Σ-nodes of t to a Boolean. Let $dom(\beta) \subseteq nodes_\Sigma(t)$ be the domain of β. For instance, the annotation $\beta = [1 \mapsto 1, 2 \mapsto 0]$ maps node 1 to 1 and node 2 to 0. We denote by $t * \beta$ the annotated tree obtained from t by relabeling all nodes ν in the domain of β to $(t[\nu], \beta(\nu))$

while preserving the labels of all other nodes. Let Q be a query with schema D. We call an annotation β for t *Q-consistent* if all nodes $\nu \in dom(\beta)$ satisfy:

$$\beta(\nu) = 1 \Leftrightarrow \nu \in Q(t).$$

In this case, we call $t * \beta$ an *annotated example for Q*. Note that annotations may be partial. Note also that 0-annotations are strict in that all nodes annotated by 0 in some annotated example for Q must be rejected by Q.

We will need a projection operation on annotated trees which deletes all annotations. The Σ-projection of a language L of annotated trees is the language $\Pi_\Sigma(L)$ of pruned trees $t \in \mathcal{T}_\Sigma(X)$ where every t is the Σ-projection of some tree $t * \beta \in L$. For every tree automaton A with signature $\Sigma \cup (\Sigma \times \mathbb{B}) \cup X$, one can compute in linear time a nondeterministic automaton $\Pi_\Sigma(A)$ over $\Sigma \times X$ such that $\mathcal{L}(\Pi_\Sigma(A)) = \Pi_\Sigma(\mathcal{L}(A))$.

4.2 From Pruning Strategies to Pruning Functions

We next show how to lift pruning strategies in order to prune positively annotated trees. There are two main manners to do so, depending on whether one permits to prune trees with many positive annotations or trees with a single positive annotation only.

Given a pruning strategy σ, our first objective is to define a pruning function p_σ that can be applied to all positively annotated trees $t * \beta$ with $t \in \mathcal{L}(D)$, while preserving the critical regions $\sigma(t, \nu)$ of all positively annotated nodes ν with their annotations:

$$p_\sigma(t * \beta) = \left(\sqcup_{\nu \in dom(\beta)} \sigma(t, \nu)\right) * \beta$$

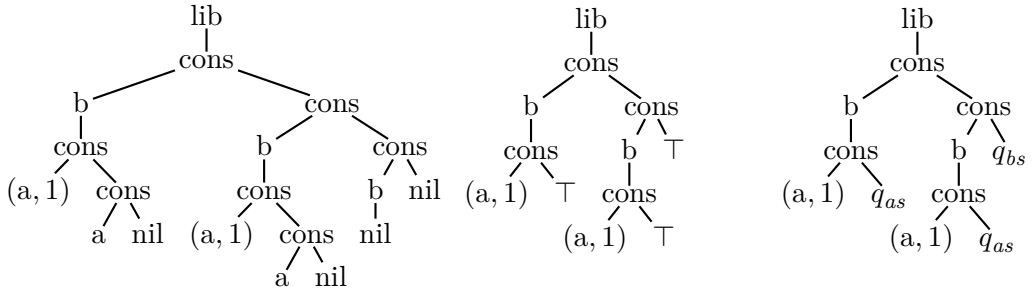
The least upper bound $\sqcup$ defines the least common instance of the σ relevant regions of all 1-annotated nodes of t. Note that this least upper bound does always exist because t is an upper bound of all $\sigma(t, \nu)$. The above least upper bound thus requires that a node is substituted by a state if and only if it is substituted by some of these prunings and not preserved by any other. Since we assume that β is a positive annotation, note that $\beta(\nu) = 1$ for all ν in the domain of β.

The concrete pruning functions $p_{path\text{-}only_D}$ and $p_{path\text{-}ext_D}$ keep all paths (respectively extended paths) to 1-annotated nodes.

Example 3 *We reconsider the library tree from Example 2 (see Figure 8). On the left of Figure 9 we present a positively annotated unpruned example for both queries Q_1 and Q_2 from Example 2. In the middle, we present the result obtained by applying function $p_{path\text{-}only}$ to this annotated example, and on the right the annotated example obtained by applying function $p_{path\text{-}only_D}$. The $p_{path\text{-}only}$ pruned example in the middle contains the minimal information relevant for query Q_1 for first-authors, while the $p_{path\text{-}only_D}$ pruned example on the right contains the minimal information relevant for query Q_2 for first-authors that are not single authors. It should be noticed that negative information will be provided to the learning algorithm independently through negatively annotated examples.*

Alternatively, we could permit to prune only trees with a single positive annotation. This is done by the following pruning function p^{can}_σ, which is defined for all $t * [\nu \mapsto 1]$ with $t \in \mathcal{L}(D)$ as below and undefined for all other annotated trees:

$$p^{can}_\sigma(t * [\nu \mapsto 1]) = \sigma(t, \nu) * [\nu \mapsto 1]$$



unpruned annotated example *path-only* pruned example *path-only*$_{\text{Lib}}$ pruned example

Figure 9: An unpruned annotated example for the two queries from Example 2 and two pruned annotated examples obtained by applying the "path-only" pruning function respectively without and with schema Lib.

If using such "canonical" pruning functions in our learning algorithm with a universal schema $D = \mathcal{U}$, we will obtain back the learning algorithm from Lemay et al. (2006) restricted to monadic queries. In our experiments, we will exclusively work with pruning functions p_σ, even though the alternative pruning function p^{can}_σ is highly promising for learning n-ary queries in particular.

In our theoretical framework, we wish to capture both cases. Therefore, we now propose a unifying definition of pruning functions by which our learning algorithm will be parameterized.

Definition 6 *A pruning function with schema D is a partial function p whose domain $dom(p)$ is a subset of positively annotated trees $t * \beta$ with $t \in \mathcal{L}(D)$ such that:*

($\mathtt{P}_1$) *every annotated tree $t * \beta \in dom(p)$ is a D-instance of $p(t * \beta)$, and*

($\mathtt{P}_2$) *for every tree $t \in \mathcal{L}(D)$ with node ν, $p(t * [\nu \mapsto 1])$ is defined and preserves node ν with its label.*

Lemma 7 *For any pruning strategy σ, both p_σ and p^{can}_σ are pruning functions.*

The proof is straightforward. It should also be noticed that pruning functions need to be adapted to unranked trees, before they become suitable for our experiments on XML query induction. Any pruning function on unranked trees can be compiled back to a (more involved) pruning function on ranked trees via a binary encoding of unranked trees (see Section 8).

4.3 Stability for Pruning Functions

We next lift the notion of stability to pruning functions. Let $t * \beta$ be an annotated example for query Q with schema D and p a pruning function with the same schema. We call t_1 a *(β, p)-variant of a tree t* if t_1 is a D- completion of the unique tree t'_1 such that $t'_1 * \beta' = p(t * \beta)$ for some β'. Note that β provides positive annotations only given that p is a pruning function. Furthermore, β' must be the restriction of β to $nodes_\Sigma(t_1)$. The nodes of $dom(\beta')$ are called determined by the (β, p)-variant t_1 of t.

Definition 8 *Let D be a deterministic tree automaton, p be a pruning function and Q be a query both with schema D. We say that Q is p-stable if for any annotated example $t * \beta$ for Q in $dom(p)$, and any node ν determined by any (β, p)-variant t_1 of t satisfies $\nu \in Q(t_1)$.*

Proposition 9 *Let σ be a pruning strategy and Q a query with the same schema D, then:*

$$Q \text{ is } \sigma\text{-stable} \Leftrightarrow Q \text{ is } p_\sigma\text{-stable.}$$

Proof "$\Rightarrow$". Let $t * \beta \in dom(p_\sigma)$ be an annotated example for Q and t' a (β, p_σ)-variant of t that determines ν. Since β is Q-consistent it follows that $\nu \in Q(t)$. We have to show that $\nu \in Q(t')$. By definition of p_σ, $t' = \sqcup_{\nu' \in dom(\beta)} \sigma(t, \nu')$, so that t' is a D-completion of $\sigma(t, \nu)$. The σ-stability of Q then implies that $\nu \in Q(t')$ as required.

"$\Leftarrow$". Let $\nu \in Q(t)$ and t_1 a D-completion of $\sigma(t, \nu)$. We have to show that $\nu \in Q(t_1)$. The annotation $\beta = [\nu \mapsto 1]$ is a Q-consistent for t and satisfies $\sigma(t, \nu) * \beta = p_\sigma(t * \beta)$. Hence, t_1 is a (β, p_σ)-variant of t that determines ν. The p_σ-stability of Q yields that $\nu \in Q(t_1)$. ■

4.4 Stability Characterization

We now characterize p-stable queries by languages of p-pruned positively annotated trees. Let D be a deterministic automaton and p be a pruning function with schema D. For any query Q with schema D, let $\mathcal{L}_Q$

$$\mathcal{L}_Q = \{t * \beta \mid \beta \text{ is a } Q\text{-consistent annotation of } t \in \mathcal{L}(D)\}$$

be the set of unpruned annotated examples for Q. Note that users of a query induction system will provide elements of $\mathcal{L}_Q$ for the target query Q. Then, the set of pruned annotated examples $p(\mathcal{L}_Q)$ is the set of all p-images of positively annotated examples for Q that belong to $dom(p)$. Note that in our XML information extraction tool, example trees will contain both positive annotations and negative annotations. In a preprocessing step, we collect positively annotated examples to which the pruning function will be applied. We will also collect negatively annotated examples which must remain unpruned in contrast.

Conversely, every language of pruned annotated trees defines a query. Let L be a language of annotated trees, it defines a query with domain $\mathcal{L}(D)$ that we denote by $\mathcal{Q}_L$ such that for all trees t in $\mathcal{L}(D)$:

$$\mathcal{Q}_L(t) = \{\nu \mid \exists t' * \beta \in L \text{ such that } \beta(\nu) = 1 \text{ and } t \in compl_D(t')\}$$

The following proposition shows that stable queries can be uniquely identified by their language of pruned positively annotated trees, under the assumption that the schema is known.

Proposition 10 *Any p-stable query Q is defined by the language $L = p(\mathcal{L}_Q)$, that is $\mathcal{Q}_L = Q$.*

Proof We assume that Q is p-stable and show that $\mathcal{Q}_L = Q$. Both queries have the same domain $\mathcal{L}(D)$, so it is sufficient to show that, for all $t \in \mathcal{L}(D)$ that $\mathcal{Q}_L(t) = Q(t)$.

"$\supseteq$" Assume $\nu \in Q(t)$. Then $\beta = [\nu \mapsto 1]$ is a Q-consistent annotation of t, so that by $(\mathtt{P}_2)$, the tree $t * \beta$ can be pruned by p while preserving ν with its label. By definition of $L = p(\mathcal{L}_Q)$, we have $p(t * \beta) \in L$. Let $t' * \beta'$ be equal to $p(t * \beta)$. Since ν is the only node ν in the domain of β and is preserved, we can deduce that $\beta' = \beta$, so that $t' * \beta \in L$. Furthermore, $t \in \mathit{compl}_D(t')$ as a consequence of condition $(\mathtt{P}_1)$ on pruning functions. Therefore, $\nu \in \mathcal{Q}_L(t)$ by definition of $\mathcal{Q}_L$.

"$\subseteq$" Assume $\nu \in \mathcal{Q}_L(t)$. Then, by definition of $\mathcal{Q}_L$, there exists $t' * \beta \in L$ such that $t \in \mathit{compl}_D(t')$ and $\beta(\nu) = 1$. Now, by definition of $L = p(\mathcal{L}_Q)$, there exists an annotated example $t_1 * \beta_1$ for Q such that $t' * \beta = p(t_1 * \beta_1)$. We have $\beta_1(\nu) = 1$ by condition $(\mathtt{P}_1)$ in the definition of pruning functions, thus $\nu \in Q(t_1)$ by definition of Q-consistency. Finally, notice that t is a (β_1, p)-variant of t_1 that determines ν. Therefore $\nu \in Q(t)$ follows from p-stability. ■

Theorem 11 *Let D be a deterministic tree automaton, p a pruning function and Q a query both with schema D. The following two properties are then equivalent:*

1. *Query Q is p-stable.*
2. *Language $L = p(\mathcal{L}_Q)$ defines query Q in that $Q = \mathcal{Q}_L$.*

Proof The implication "1 $\Rightarrow$ 2" was shown by Proposition 10, so it remains to prove "2 $\Rightarrow$ 1". We assume $\mathcal{Q}_L = Q$ and show that Q is p-stable. Let $t * \beta$ be a Q-consistently annotated tree in $\mathit{dom}(p)$ and t_1 a (β, p)-variant of t determining ν. Variant t_1 is then a D-completion of t_1' where $t_1' * \beta' = p(t * \beta)$ for some β'. The Q-consistency of β on t implies that $t_1' * \beta' \in L$. Furthermore, it implies that $\nu \in Q(t)$ since β must be all positive by definition of pruning functions. Since variant t_1 determines ν, it follows that $\beta'(\nu) = 1$. Hence $\nu \in \mathcal{Q}_L(t_1)$ by definition of $\mathcal{Q}_L$. With our assumption $\mathcal{Q}_L = Q$, we can conclude that $\nu \in Q(t_1)$ as required. ■

5. Regularity

From the view point of XML information extraction, XPath queries with DTD schema restrictions are of highest interest for query induction. Modulo binary encoding of unranked trees, DTD schemas can be expressed by deterministic tree automata. Furthermore, since we ignore data values, XPath queries can be defined by tree automata too, that operate on binary encodings of unranked trees. This motivates our study of the class of regular queries for query induction.

In our learning algorithm we will represent regular p-stable queries by tree automata that recognize the language $p(\mathcal{L}_Q)$ of pruned positively annotated examples for Q. The objective of this section is to show that every regular p-stable query can be represented in this manner under the assumption that the pruning function p is regular too, a notion that we will introduce.

5.1 Regular Stable Queries

We recall a definition of regular queries and show how to represent stable regular queries by regular languages of pruned annotated examples.

Definition 12 *A query Q is regular if the set $\mathcal{L}_Q$ of unpruned annotated examples for Q is a regular tree language.*

As before, let D be a fixed deterministic tree automaton with signature Σ and state set X, so that pruned annotated tree have the signature $\Sigma \cup (\Sigma \times \mathbb{B}) \cup X$. Given a tree automaton A that recognizes pruned annotated trees, we define the query $\mathcal{Q}_A$ with schema D by $\mathcal{Q}_{\mathcal{L}(A)}$. Recall that our notation leaves the dependence of D implicit.

Query answering for regular queries is an important task for every interactive query induction system. Fortunately, queries defined by tree automata can be answered efficiently. Indeed, for every tree t in $\mathcal{L}(D)$, the set $\mathcal{Q}_A(t)$ of query answers can be computed in time $O(|A|\ |D|\ |t|)$ even without determinism. This result is folklore. It can be shown, for instance, by converting (A, D) into a monadic Datalog program of size $O(|A|\ |D|)$ that defines the same query and applying the algorithms for monadic Datalog from Gottlob and Koch (2002).

Lemma 13 *A query Q with schema D is regular if and only if $Q = \mathcal{Q}_L$ for some regular language L of D-pruned annotated trees.*

Proof For the one direction, note that $\mathcal{Q}_{\mathcal{L}_Q} = Q$, so that we can choose $L = \mathcal{L}_Q$ if $\mathcal{L}_Q$ is regular. The other direction is more tedious. Given automata A and D we have to construct an automaton recognizing $\mathcal{L}_{\mathcal{Q}_A}$ depending on D. How this can be done is shown in Appendix A. ■

It should be noticed that the automaton construction in the above proof requires exponential time in the size of A. Fortunately, this construction will only be used to clarify the expressiveness of our query representation formalism in Theorem 18. It will not be used by our query induction algorithm.

Proposition 14 *A p-stable query Q is regular if its language $p(\mathcal{L}_Q)$ is regular.*

Proof Let Q be a p-stable query and $L = p(\mathcal{L}_Q)$. Proposition 10 shows that $Q = Q_L$. Lemma 13 thus implies that Q_L is regular, under the assumption that L is regular. ■

The converse holds only for regular pruning functions introduced next (see Theorem 18).

5.2 Regular Pruning Functions

We will also need a formalism for specifying pruning functions. Again, we will use tree automata for this purpose, leading to the notion of regular pruning functions.

Example 4 *We reconsider the example on pruned libraries for the pruning function obtained from the path-only$_{Lib}$ strategy. All maximal subtrees in which no node is annotated by* 1

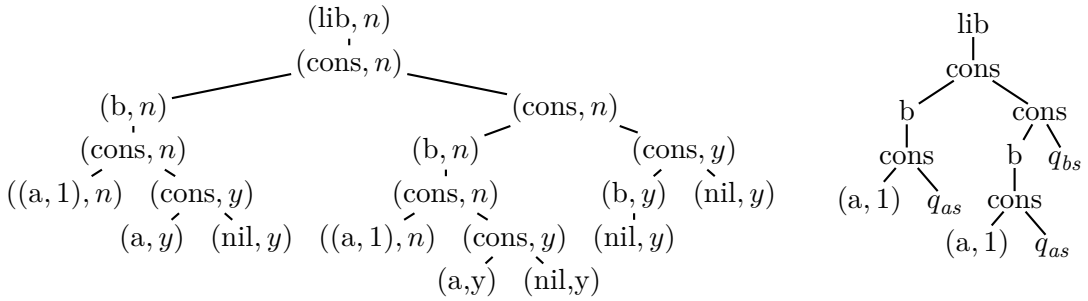


Figure 10: The unpruned annotated tree of Figure 9 with nodes annotated with the symbols y and n according to "path-only" pruning functions is on the left. Its pruning according to pruning function $p_{\mathit{path\text{-}only}_{\mathrm{Lib}}}$ is shown on the right.

must be pruned and all nodes above nodes annotated by 1 *must be preserved. This can be done by a finite state machine. For this, we will annotate nodes that are to be pruned by* y *and nodes that must be preserved by* n*. For instance, the annotation of the tree of Figure 9 is shown in Figure 10. Then, it is easy to define a tree automaton which checks whether the definition of a "path-only" pruning function is satisfied. This automaton can also be used to apply the pruning function.*

Formally, let us consider a pruning function p and an annotated tree $t * \beta$ in its domain. We call a node ν of $t * \beta$ unpruned if it belongs to $\mathit{nodes}_{\Sigma\cup\Sigma\times\mathbb{B}}(p(t * \beta))$ and pruned otherwise. Let $t * \beta * p$ be the tree obtained from $t * \beta$ by annotating all nodes that were pruned by p with y and all others by n. This way we can identify p with the language $\mathcal{L}_p = \{t * \beta * p \mid t * \beta \in \mathit{dom}(p)\}$. Note that $\mathcal{L}_p$ contains unpruned trees only. We say that a tree automaton P with signature $(\Sigma \cup (\Sigma \times \mathbb{B})) \times \{y, n\}$ defines a pruning function p with schema D if $\mathcal{L}_p = \{s \in \mathcal{L}(P) \mid \Pi_\Sigma(s) \in \mathcal{L}(D)\}$. Note that schema validation for D is considered as an external issue for P. This pruning function p is then denoted by $\wp_{P,D}$.

Definition 15 *A pruning function* p *with schema* D *is called* regular *if it is equal to* $\wp_{P,D}$ *for some tree automaton* P*.*

For instance, the pruning function $p_{\mathit{path\text{-}only}_D}$ is regular. It can be defined by the same automaton with 2 states for all D (since schema validation is done externally). This automaton checks whether a node has a 1-annotated descendant (state 1) or not (state 0). Nodes in state 1 must be labeled by n and nodes in state 0 by y. Both states are final. For all symbols $f^{(k)}$ in Σ where $k \geq 0$ we define the following rules, where each occurrence of symbol $*$ stands for either state 0 and 1.

$$(f, y)(0, \ldots, 0) \to 0 \quad ((f, 1), n)(*, \ldots, *) \to 1 \quad (f, n)(*, \ldots, *, 1, *, \ldots *) \to 1$$

Pruning function $p_{\mathit{path\text{-}ext}_D}$ is also regular. It is sufficient to add a further state to the previous automaton, which checks whether a node is a sibling of a positively annotated node (or not). We leave the precise automaton construction to the reader.

Any pair (P, D) can be transformed in polynomial time into a linear bottom-up tree transducer (Comon et al., 2007) that defines the same pruning function $\wp_{P,D}$. Such transducers, however, may become nondeterministic even if P and D are deterministic, as for instance for $p_{\textit{path-ext}_D}$. Furthermore, if we avoid such a conversion, we can indeed evaluate the application of a pruning functions to an annotated tree efficiently, as we show next.

Lemma 16 *Let P and D be deterministic tree automata, $p = \wp_{P,D}$, and $t * \beta$ an unpruned annotated tree. One can decide whether $t * \beta$ belongs to $dom(p)$ and in this case compute $p(t * \beta)$ in time $O(|P|\ |t| + |D| + |t|)$.*

Proof We compute the projection automaton $P' = \Pi_{\Sigma \cup (\Sigma \times \mathbb{B})}(P)$ by "integrating" the labels y and n into the states of P. Note that P' may be nondeterministic despite of the determinism of P. Let $t * \beta$ be an unpruned annotated input tree. Since D is deterministic we can decide in time $O(|D| + |t|)$ whether $t \in \mathcal{L}(D)$. If not, we return that $t * \beta$ does not belong to $dom(p)$. Otherwise, there exists at most one unpruned tree $s \in \mathcal{L}(P)$ such that $\Pi_{\Sigma \cup \Sigma \times \mathbb{B}}(s) = t * \beta$, since p is a partial function satisfying $\mathcal{L}_p = \{s \in \mathcal{L}(P) \mid \Pi_\Sigma(s) \in \mathcal{L}(D)\}$. Therefore, and since P is deterministic, P' may have at most one successful run on $t * \beta$ and possibly many unsuccessful runs. This unique successful run would contain all information on whether a node is to be pruned or not in the $\{y, n\}$ component of its state. It is thus sufficient to compute the successful run of P' on $t * \beta$ if it exists and to detect is nonexistence otherwise. This can be done by running the nondeterministic automaton P' on t in time $O(|P'|\ |t|)$ in a bottom up phase, then testing whether a final state got reached, and if yes selecting the successful run in a top-down phase. Since P' contains at most twice as many states as P, its size is linear in that that of P. The overall computation time is thus in $O(|P|\ |t| + |D| + |t|)$. ■

Lemma 17 *Let D, A, and P be deterministic tree automata. If $L = \mathcal{L}(A)$ is a language of unpruned annotated trees and $p = \wp_{P,D}$ a pruning function then $p(L)$ can be recognized by a deterministic tree automaton of size $O(|D|\ 2^{|P|}\ |A|)$.*

Proof Let $p = \wp_{P,D}$ and $L = \mathcal{L}(A)$ for deterministic tree automata P, D, A. We rely on a similar algorithm as for computing $p(t * \beta)$ from $t * \beta$ except that we must deal with states from X in pruned trees $p(t * \beta)$. So we use the projection automaton $P' = \Pi_{\Sigma \cup (\Sigma \times \mathbb{B})}(P)$ but we add rules $q \to (r, y)$ for all states $q \in X$ and r of P. Then, we determinize automaton P', lift D to the automaton D' on $\Sigma \cup (\Sigma \times \mathbb{B}) \cup X$ that runs D on the Σ component of any pruned input trees, while adding rules $q \to q$ for all $q \in X$. We note that the determinism of D inherits to D'. We then compute the product $P' \cap D' \cap A$. By running this product on a pruned input tree $t * \beta$, we can test whether there exists a D-completion $t' * \beta'$ such that $t' \in \mathcal{L}(D)$ and $p(t' * \beta') = t * \beta$. The product automaton is deterministic, recognizes $p(L)$, and is of size $O(|D|\ 2^{|P|}\ |A|)$. ■

Theorem 18 *Let Q be a p-stable query for a regular pruning function p. Then:*

$$Q \textit{ is regular} \Leftrightarrow p(\mathcal{L}_Q) \textit{ is regular.}$$

Proof If Q is regular then $\mathcal{L}_Q$ is regular, and thus $p(\mathcal{L}_Q)$ by Lemma 17. The converse implication was shown in Proposition 14. ■

This shows that every regular p-stable query Q can be represented by a (minimal deterministic) tree automaton that recognizes the language $p(\mathcal{L}_Q)$ of p-pruned positively annotated examples for Q. This is the representation on which we will base our learning algorithm for regular p-stable queries.

5.3 Deciding Stability

For our experimental validation, it will be necessary to decide whether a regular query is stable for a regular pruning function. This can be done in polynomial time:

Theorem 19 *Let D and P be deterministic automata defining a regular pruning function $p = \wp_{P,D}$ and Q a query with domain D. For any tree automaton A recognizing $\mathcal{L}_Q$ we can decide in time $O(|A|^2\ |D|\ |P|)$ whether Q is p-stable.*

The quite technical proof which is based on so called recognizable relations between trees (which is based on the idea of regular language of overlays of tree tuples, see e.g. Comon et al. (2007)) is deferred to Appendix A.

6. Algorithms for Consistency Checking

In this section, we will present algorithms for checking consistency properties for regular sets of annotated examples that we will introduce: D-functionality and p-consistency. These consistency checks will be the prime subroutines of our learning algorithm. They help us to avoid the need for further negative examples for the target language, beside the negatively annotated examples for the target query.

6.1 D-Functionality

We consider languages of pruned annotated trees with mixed positive and negative annotations. D-functionality is a consistency notion for such sets which requires that positive and negative annotations are non-contradictory.

Definition 20 *We call a language $L \subseteq \mathcal{T}_{\Sigma\cup(\Sigma\times\mathbb{B})}(X)$ of annotated trees* D-functional *if for any $t * \beta, t' * \beta' \in L$, if t and t' are compatible with respect to D then β and β' coincide on $dom(\beta) \cap dom(\beta')$.*

The language $\{(a,1)(\top),(a,0)(\top)\}$, for instance, is not $\mathcal{U}$-functional. We next relate p-stable queries to D-functional languages.

Proposition 21 *If Q is a p-stable query with schema D, then the language $p(\mathcal{L}_Q) \cup \mathcal{L}_Q$ is D-functional.*

Proof The proof is by contradiction. Let us assume that $p(\mathcal{L}_Q) \cup \mathcal{L}_Q$ is not D-functional. Since domains of pruning functions contain positively annotated trees, there must exist

a p-pruned example $t'_1 * \beta'_1 \in p(\mathcal{L}_Q)$ for Q and an unpruned example $t_2 * \beta_2 \in \mathcal{L}_Q$ for Q such that t'_1 and t_2 are compatible with respect to D, so that there exists a node ν with $\beta'_1(\nu) = 1$ and $\beta_2(\nu) = 0$. By definition of $p(\mathcal{L}_Q)$, there exists an unpruned example $t_1 * \beta_1$ for Q such that $t'_1 * \beta'_1 = p(t_1 * \beta_1)$. Property ($\mathtt{P}_1$) of pruning functions implies that $\beta_1(\nu) = \beta'_1(\nu) = 1$. Since β_1 is a Q-consistent annotation of t_1, it follows that $\nu \in Q(t_1)$, and since β_2 is a Q-consistent annotation of t_2 that $\nu \notin Q(t_2)$. Compatibility implies that t_2 is a D-completion of t'_1. Hence, t_2 is a (β_1, p)-variant of t_1 that determines ν. The p-stability of Q yields $\nu \in Q(t_2)$. Contradiction. ■

We now show that D-functionality can be tested efficiently for regular queries Q_A, by reduction to testing emptiness of tree automata. Note that determinism does not help to improve emptiness tests, so that it will not help to assume that A is deterministic here.

Theorem 22 *Let D be a deterministic tree automaton with signature Σ and state set X, and A be a tree automaton with signature $\Sigma \cup (\Sigma \times \mathbb{B}) \cup X$. Whether $\mathcal{L}(A)$ is D-functional can be decided in time $O(|D|\ |A|^2)$.*

Proof We write $s_1 \circledast s_2$ for the overlay of trees s_1 and s_2, where missing nodes are filled up with a fresh $\perp$-symbol. Let us consider the language L_{contra} of overlays $(t_1 * \beta_1) \circledast (t_2 * \beta_2)$ such that $t_1 * \beta_1$ and $t_2 * \beta_2$ are accepted by A, that t_1 and t_2 have a common completion i.e. $compl_D(t_1) \cap compl_D(t_2) \neq \emptyset$, and that there exists a node ν with contradicting annotations $\beta_1(\nu) \neq \beta_2(\nu)$. By definition of D-functionality, $\mathcal{L}(A)$ is D-functional if and only if $L_{contra} = \emptyset$.

The existence of a common D-completion can be checked by running a single copy of D jointly on t_1 and t_2, while checking that the states of D occurring in t_1 and t_2 are chosen appropriately. More precisely, if a node in $t_1 \circledast t_2$ is labeled by $(f, \perp)$ then D runs on the left component and if it is labeled with $(\perp, f)$ then it is run on the right component. Nodes with labels (f, q), (q, f) are correct, if the state reached by D is equal to q. From this time point on, D continues on both components while testing their equality. Labels (q_1, q_2) with different state contradicts the compatibility of t_1 and t_2 (since D is deterministic), as well as labels (f_1, f_2) with different function symbols. We have also to test that the joined run of D on t_1 and t_2 does reach a final state.

Membership of $t_1 * \beta_1$ and $t_2 * \beta_2$ to $\mathcal{L}(A)$ can be checked by running two copies of A in parallel on their overlay. Last but not least, the existence of a contradiction between annotations can be checked by a very simple automaton with a single state. Thus, L_{contra} can be recognized by a product automaton of size $O(|D|\ |A|^2)$. As emptiness is testable in linear time for tree automata, the theorem follows. ■

Note that if one can assume that the Σ-projections of all trees recognized by A satisfy schema D, as we can do for our learning algorithm, then the test in the proof can be simplified, since never D has to be run in parallel with both copies of A. In this case, D-functionality can be checked in time $O(|D|\ |A| + |A|^2)$.

6.2 p-Consistency

In our learning algorithms, we will need to check whether a language of annotated trees contains only p-pruned trees for a given regular pruning function p.

Definition 23 *We call a language L of annotated trees* p-consistent *if all trees in L are p-pruned, i.e., if $L \subseteq p(\mathcal{T}_{\Sigma\cup(\Sigma\times\mathbb{B})})$.*

Proposition 24 *Let $p = \wp_{P,D}$ be a regular pruning function, let $L = \mathcal{L}(A)$ be a regular set of pruned annotated trees defined by a deterministic automaton with signature $\Sigma\cup(\Sigma\times\mathbb{B})\cup X$. Whether $\mathcal{L}(A)$ is p-consistent can be decided in time $O(|A|\ |D|\ 2^{|P|})$.*

Proof By definition of p-consistency, we must test whether $\mathcal{L}(A) \subseteq p(\mathcal{T}_{\Sigma\cup(\Sigma\times\mathbb{B})})$. By Lemma 13, we can construct a deterministic tree automaton recognizing $p(\mathcal{T}_{\Sigma\cup(\Sigma\times\mathbb{B})})$ in time $O(|D|\ 2^{|P|})$. We can thus decide p-consistency by inclusion checking. According to Proposition 1, inclusion can be checked in time $O(|A|\ |D|\ 2^{|P|})$. ■

The p-consistency test may lead to an exponential blow-up in the size of P due to determinization of the projection of P. In practice this does not raise any problems. The first reason is that the usual determinization procedure does often behaves much better than in the worst case. The second reason is that the automaton P defining the pruning function will usually be very small. For instance, "path-only" pruning functions can be defined by an automaton with 2-states (indeed the same automaton for all schemas D), and "path-extended" pruning functions with 3-states. Pruning functions derived from the unranked case via the binary encoding will be defined with no more than 5-states.

7. Learning Stable Regular Queries

In this section, we present a learning algorithm for stable queries and prove a formal learnability result.

7.1 Learning from p-Pruned Samples

We present a learning algorithm that infers p-stable queries from p-pruned samples and show that it satisfies the learning model from polynomial time and data. The nontrivial aspect is that p-pruned samples are indeed sufficient.

We suppose that the schema is fixed by a deterministic automaton D, and that the pruning function $p = \wp_{P,D}$ is fixed by a deterministic tree automaton P. As shown before, a p-stable regular query Q is uniquely defined by the language $L = p(\mathcal{L}_Q)$ of all p-pruned examples for Q. The idea is therefore to identify the minimal deterministic tree automaton for the language $L = p(\mathcal{L}_Q)$ associated with the p-stable target query Q. As input, it will receive a p-pruned sample for the target query.

Definition 25 *A p-pruned sample is a pair (S^+, S^-) where S^+ is a p-consistent finite set of positively annotated trees and S^- a finite set of negatively annotated trees such that $S^+ \cup S^-$ is D-functional.*

Note that only positively annotated examples can be p-pruned, since pruning functions do not apply to examples with negative annotations. We can now state our main result of the learnability of p-stable queries from p-pruned samples.

Theorem 26 *Let schema D be a deterministic tree automaton and p be a fixed regular pruning function with schema D. Let $\mathcal{A}$ be the class of deterministic tree automata that recognize languages $p(\mathcal{L}_Q)$ of some regular p-stable query Q and $\mathcal{S}$ the class of p-pruned samples of annotated examples. Then, the class of p-stable regular queries represented by automata in class $\mathcal{A}$ is learnable in polynomial time with polynomially many examples from samples in the class $\mathcal{S}$. I.e. any automaton A in $\mathcal{A}$ has a characteristic sample $char(A)$ in $\mathcal{S}$ whose cardinality is polynomial in the size of A and there is an algorithm p-stable-$\mathcal{RPNI}_D$ such that, for any sample S' in $\mathcal{S}$ that subsumes the characteristic sample $char(A)$ of an automaton A recognizing the language $L = p(\mathcal{L}_Q)$ of a p-stable query Q, algorithm p-stable-$\mathcal{RPNI}_D$ outputs in polynomial time in the size of S' the unique minimal deterministic tree automaton recognizing L.*

Proof The proof is by reduction to the problem of learning regular tree languages represented by deterministic tree automata from positive and negative examples (García and Oncina, 1993). Let $\mathcal{A}'$ be the class of deterministic tree automata with signature $\Sigma' = \Sigma \cup (\Sigma \times \mathbb{B}) \cup X$ and $\mathcal{S}'$ the class of samples (S'^+, S'^-) of positive and negative examples $S'^+, S'^- \subseteq \mathcal{T}_{\Sigma'}$ with $S'^+ \cap S'^- = \emptyset$. From the learnability result for deterministic tree automata, we know that every automaton A in $\mathcal{A}'$ has a characteristic sample $char'(A) = (S'^+, S'^-)$ in $\mathcal{S}'$ with $S'^+ \subseteq \mathcal{L}(A)$ and $S'^- \cap \mathcal{L}(A) = \emptyset$ whose cardinality is polynomial in the size of A. Also, there is an algorithm $\mathcal{RPNI}$ such that, for every sample S in $\mathcal{S}'$ that subsumes the characteristic sample $char'(A)$, $\mathcal{RPNI}$ with input S outputs in polynomial time in the size of S the unique minimal deterministic tree automaton recognizing $\mathcal{L}(A)$.

In the first step of the proof, we construct the characteristic sample $char(A) = (S^+, S^-)$ in $\mathcal{S}$ for A in $\mathcal{A}$. We fix a total order on trees over Σ, such that trees with fewer nodes become smaller. Let A be a deterministic tree automaton recognizing the language $L = p(\mathcal{L}_Q)$ of the p-stable target query Q. Since automaton A belongs to class $\mathcal{A}'$, there exists a characteristic sample $char'(A) = (S'^+, S'^-)$ in $\mathcal{S}'$. We define $char(A)$ in $\mathcal{S}$ from $char'(A)$ as follows. All trees in S'^+ belong to $L = p(\mathcal{L}_Q)$, so we put them into S^+. For all trees s in S'^- we proceed as follows:

- If s does not belong to the image of p then we can safely ignore s, since our algorithm will always check that the language of the target automaton will be p-consistent (recall that p is fixed and known by the algorithm).
- Otherwise, let t be the least tree in the total order fixed above such that $s = p(t * \beta)$ for some β, and fix one of those βs. By definition of pruning functions, β must be a positive annotation. Since $s \in S'^-$, however, β cannot be consistent with Q, so there exists a node $\nu \notin Q(t)$ such that $\beta(t) = 1$. We add $t * [\nu \mapsto 0]$ to S^-.

We obtain a sample $char(A) = (S^+, S^-)$ whose cardinality is at most linear in the cardinality of $char'(A)$, and thus polynomial in the size of A. Moreover, it follows from Proposition 21 that $S^+ \cup S^-$ is D-functional since Q is p-stable. Therefore, $char(A)$ is a p-pruned sample in the class $\mathcal{S}$.

```
fun p-stable-RPNI_D(S+, S−) // input a p−pruned sample (S+, S−)
  A ← init(S+) // deterministic tree automaton recognizing S+ such
               // that at most one tree is recognized per state
  let (q1, ..., qn) = sort states of A consistently with order on trees
  Ok ← ∅ // states already merged
  for i=1 to n do
    if qi ∈ Ok then skip else
      for j=1 to i−1 do
        let A' = det-merge(A, qi, qj) // qi becomes a reference to qj
        if L(A') ∪ S− is D−functional // Theorem 22
           and L(A') is p−consistent // Proposition 24
        then
          A ← A'
          update Ok by adding newly merged states from computing A'
          exit // inner for loop
        else
          skip
      end // inner for loop
      add qi to Ok // if qi got merged then it belonged already to Ok
  end // outer for loop
  output A // a deterministic tree automaton defining query Q_A
```

Figure 11: Learning algorithm for p-stable queries from p-pruned examples.

In Figure 11, we define our learning algorithm *p-stable-RPNI*$_D$ which receives p-pruned samples (S^+, S^-) in $\mathcal{S}$ as input. It is parameterized by a deterministic tree automaton D and a regular pruning function p with schema D. It remains to show for every sample (S^+, S^-) in $\mathcal{S}$ that subsumes the characteristic sample $char(A)$ of an automaton A recognizing the language $L = p(\mathcal{L}_Q)$ of a p-stable query Q, that *p-stable-RPNI*$_D(S^+, S^-)$ is the unique minimal deterministic tree automaton recognizing L and that it is computed in polynomial time.

For this, let $A \in \mathcal{A}$ be an automaton. We consider $char(A)$ as defined before from the characteristic sample $char'(A)$. We first show that:

$$\textit{p-stable-RPNI}_D(char(A)) = \textit{RPNI}(char'(A)).$$

Before showing this, we recall the principles of *RPNI* and explain the differences to *p-stable-RPNI*$_D$. Algorithm *RPNI* receives as input a finite sample of positive and negative examples for the target language. At the beginning it computes an initial automaton $init(S^+)$ recognizing the positive examples S^+. Each of its states can be associated with the least subtree that is evaluated to this state. Thereby, states become totally ordered, as well as pairs of states. State merge operations preserving determinism are then tried out in this order. A state merge operation is accepted if it preserves consistency in that no negative example can be recognized. Otherwise, it is rejected and must be undone. This procedure is repeated exhaustively. Algorithm *p-stable-RPNI*$_D$ is similar except that it receives a p-pruned sample as input, and that the consistency test is replaced by a test for D-functionality and p-consistency.

Let $char(A) = (S^+, S^-)$ be the p-pruned sample constructed from $char'(A) = (S'^+, S'^-)$ as described above. By construction, $S'^+ \subseteq S^+$. Therefore, it follows from the learn-

ability result for deterministic tree automata that $\mathcal{RPNI}(S^+, S'^-) = \mathcal{RPNI}(char'(A))$ is the unique minimal deterministic automaton recognizing $\mathcal{L}(A)$. It remains to show that $p\text{-}stable\text{-}\mathcal{RPNI}_D(S^+, S^-) = \mathcal{RPNI}(S^+, S'^-)$. Both algorithms start by constructing the initial automaton $init(S^+)$. Therefore, along their computations, both algorithms will try to perform the same state merge operations in the same order. Successful attempts will be accepted, while all others with be rejected and undone. It remains to show that both algorithms accept the same state merge operations. A merge operation is rejected by $\mathcal{RPNI}$ if the automaton obtained recognizes some tree $s \in S'^-$. This translates either to a lack of p-consistency of the current automaton A reached by evaluating $p\text{-}stable\text{-}\mathcal{RPNI}_D$ or to a lack of D-functionality of $\mathcal{L}(A) \cup S^-$. Conversely, if p-consistency or D-functionality fail for the current automaton of $p\text{-}stable\text{-}\mathcal{RPNI}_D$, then the computation of $\mathcal{RPNI}$ cannot accept this merge operation either, since otherwise its final automaton would accept a larger language than the target language. Thus, $p\text{-}stable\text{-}\mathcal{RPNI}_D(S^+, S^-) = \mathcal{RPNI}(S^+, S'^-)$.

The computation time for $\mathcal{RPNI}$ is polynomial in the size of the input sample. The additional tests for D-functionality and p-consistency performed by $p\text{-}stable\text{-}\mathcal{RPNI}_D$ are in polynomial time (for fixed p) by Theorem 22 and Proposition 24. Thus, the overall computation time of $p\text{-}stable\text{-}\mathcal{RPNI}_D$ is polynomial in the size of the input sample. It can also be proven that if $p\text{-}stable\text{-}\mathcal{RPNI}_D$ receives a superset of $char(A)$ as input, then the output is again the minimal deterministic automaton recognizing the language $L = p(\mathcal{L}_Q)$ of the target p-stable query Q. ■

As shown above, automaton $p\text{-}stable\text{-}\mathcal{RPNI}_D(S^+, S^-)$ can be computed in polynomial time depending on the size of the input sample (S^+, S^-), for fixed regular pruning function $p = \wp_{P,D}$. This may change if moving the deterministic automaton P to the inputs, since then there may be an exponential blow up in the size of P in the worst case (see Proposition 24). Therefore, the choice of the pruning function requires a little care: they should be defined by automata with few states only.

The performance of the inclusion test for checking p-consistency is crucial for practical efficiency, since it is performed repeatedly at every merge attempt of learning algorithm $p\text{-}stable\text{-}\mathcal{RPNI}_D$. Therefore, we have implemented our inclusion test such that it is incremental with respect to the addition of rules to automaton A defining the query hypothesis, see (Champavère et al., 2009).

We also designed and implemented an alternative algorithm, where schema consistency is ensured by static state-typing, as in previous inference algorithms for regular languages (Coste et al., 2004; Oncina and Varó, 1996). The experimental results were not convincing though: the algorithm worked well only for queries whose automata share much of their "structure" with the schema, which is rarely the case in our applications. Furthermore, state-typing algorithms are not yet well-founded theoretically, i.e., no result on learning in the limit exists.

7.2 Learning from Unpruned Samples

In our experiments, a user annotates unpruned samples for the target query, which may or may not be stable for a given pruning strategy σ. We next discuss how to produce p_σ-pruned

```
fun σ-stable-learn_D(S) // input a sample of unpruned annotated trees
                        // that is functional and consistent with schema D
  let S⁺ = p_σ(S^=1)  // remove 0–annotations in S and apply p
  let S⁻ = S^=0       // remove 1–annotations in S
  if S⁺ ∪ S⁻ is not D–functional then raise exception 'unstable query'
     // if no exception is raised then (S⁺, S⁻) is a p–pruned sample
  output p_σ-stable-RPNI_D(S⁺, S⁻)
```

Figure 12: Learning algorithm for σ-stable queries from unpruned examples.

samples thereof, in order to obtain a learning algorithm for σ-stable queries from unpruned examples.

Definition 27 *An* unpruned sample for a schema D *is a functional tree language* $L \subseteq \mathcal{T}_{\Sigma\cup(\Sigma\times\mathbb{B})}$ *that is consistent with* D*, i.e.,* $\Pi_\Sigma(L) \subseteq \mathcal{L}(D)$. [2]

This algorithm which we call σ-*stable-learn*$_D$ is presented in Figure 12. Given an unpruned sample S for schema D, it computes a p_σ-pruned sample (S^+, S^-) if possible. The positive part S^+ is obtained by removing all negative annotations from trees in S and then applying p_σ. The negative part S^- is obtained from S by removing all positive annotations. If S is a sample for a σ-stable query, then $S^+ \cup S^-$ must be D-functional as shown by Propositions 9 and 21, so that (S^+, S^-) is indeed a p-pruned sample. In this case, p_σ-*stable-RPNI*$_D$ can be safely applied. Otherwise, the algorithm raises an exception.[3]

For instance, pruning strategy $\sigma =$ *path-only* with the universal schema $D = \mathcal{U}$, and query Q which selects all a's whose left-sibling is labeled by b. The set $S = \{f(b,(a,1)), f(a,(a,0))\}$ is an unpruned sample for Q. It induces the p_σ-pruned sample (S^+, S^-) where $S^+ = \{f(\top,(a,1))\}$ and $S^- = \{f(a,(a,0))\}$. Clearly, $S^+ \cup S^-$ is not $\mathcal{U}$-functional since Q is not σ-stable, and evaluating σ-*stable-learn*$_D(S)$ will raise exception 'unstable query'.

7.3 Selection of a Pruning Strategy

The choice of a pruning strategy has strong implications on the class of learnable queries. The more aggressive the underlying pruning strategy is, the smaller will be the class of stable queries, and thus the class of learnable queries. For more aggressive pruning strategies, however, our learning algorithm will converge more quickly. So the question is how to find appropriate pruning strategies.

We can approach the problem as follows. First we fix a finite set of pruning strategies, containing for instance *path-only*$_D$, *path-ext*$_D$, *path-only*, and *path-ext* as in our experiments. Given a sample S of unpruned trees, the idea is to choose the most aggressive of these pruning strategies, so that the evaluation of p_σ-*stable-learn*$_D(S)$ does not raise exception 'unstable query'. The inferred query that as well as the pruning strategy that is selected may still be unsatisfactory if the sample S is not sufficiently informative. In this case, a

2. A language of unpruned annotated trees is functional if and only if it is $\mathcal{U}$-functional.
3. The usage of canonical pruning functions p_σ^{can} requires some care since p_σ^{can} might be undefined for some trees in $S^{=1}$. Instead, one can add all trees $p(t * [\nu \mapsto 1])$ to S^+ such that there exists $t * \beta \in S$ with $\beta(\nu) = 1$. This approach, however, requires some algorithmic improvements to become competitive.

larger sample must be provided, so that the pruning strategy can be updated accordingly and the correct query can be found.

8. Stable Regular Queries for Unranked Trees

So far, all notions have been defined for ranked trees. But our goal is to learn XML queries over XML trees. Thus, all notions and results must be lifted to the unranked case and this is the objective of the section.

For this, we use a binary encoding. We choose the bottom-up binary encoding from Carme et al. (2004) that is also known as currying from the lambda-calculus. The advantage of using the bottom-up encoding, rather than the more frequently used top-down encoding based on first-child and next-sibling relations, is that schemas can be defined by bottom-up deterministic tree automata (rather than the less expressive top-down deterministic tree automata). For instance, any deterministic DTD for unranked trees can be transformed in polynomial time into a bottom-up deterministic tree automaton for curried binary encodings. See for instance Champavère et al. (2009) for a precise complexity analysis of the translation and further details.

Example 5 *Let us introduce currying by example. We reconsider the unranked library tree u with three books (b), of which the first two have two authors (a), and the last one none:*

$$u = lib(b(a, a), b(a, a), b)$$

Its curried encoding is as follows where the binary "application" operator @ is written in infix notation, while missing parenthesis are to be added from left to right.

$$curry(u) = lib@(b@a@a)@(b@a@a)@b$$

This way, function *curry* maps unranked trees over a label set Σ are encoded into binary trees over the ranked alphabet $\Sigma_@$ containing the binary special function symbol $@^{(2)}$ and constants for all symbols of Σ. Function *curry* is one to one and onto, that is, every binary tree over $\Sigma_@$ uniquely defines an unranked tree over Σ. However, the relationship between nodes is a little more intricate. Every node of an unranked tree u corresponds to a unique leaf node of the ranked tree $curry(u)$ and vice versa. Inner nodes of $curry(u)$, i.e., those labeled by @, do not correspond to any node of u (but to some of its edges). Node selection queries on unranked trees therefore correspond to leaf selection queries on ranked trees.

We next introduce stable regular queries and pruning strategies for unranked trees, and show how to translate them to ranked trees via a binary encoding. Let Σ be a finite label set. We denote by $\mathcal{U}_\Sigma$ the set of unranked trees over Σ. A schema will be defined as a deterministic tree automaton D with ranked signature $\Sigma_@$ and state set X. Such an automaton evaluates unranked trees by evaluating their binary encoding.

A pruned unranked tree is a unranked tree with label set $\Sigma \cup X$. By $\mathcal{U}_\Sigma(X)$ we denote the set of all pruned unranked trees. As for ranked trees, we can define a subsumption order on $\mathcal{U}_\Sigma(X)$, so that greater trees are obtained by instantiating occurrences of states by unranked trees that can be evaluated to this state by D.

An unranked pruning strategy for a schema D is a function σ that maps unranked trees $u \in \mathcal{U}_\Sigma$ with leaf nodes $\nu \in leafs(u)$ to pruned unranked trees $\sigma(u, \nu) \in \mathcal{U}_\Sigma(X)$, while

satisfying literally conditions ($\mathtt{S}_1$) and ($\mathtt{S}_2$) from the ranked case (but with u instead of t). There exists a one-to-one and onto mapping between unranked and ranked pruning strategies.

We define pruning strategies *path-only*$_D$ and *path-ext*$_D$ on unranked trees as before, such that they preserve the path to the input node and respectively the extended path. The next example shows that pruning strategies that correspond on ranked trees are quite different from what one obtains when applying the ranked *path-only*$_D$ and *path-ext*$_D$ pruning strategies to binary encodings of unranked trees. This means that the automata that defining *path-only*$_D$ and *path-ext*$_D$ need to be adapted to the unranked case appropriately.

Example 6 *Let u be the unranked library from Example 5. The unranked "path-only" pruning strategy without schema restrictions applied to the first author of the second book yields:*

$$\mathit{path\text{-}ext}(u, 2 \cdot 2) = \mathit{lib}(\top, b(a, \top), \top)$$

The correct corresponding path-only pruning of $\mathit{curry}(u)$ at the corresponding leaf $1 \cdot 2 \cdot 1 \cdot 2$ is equal to:

$$\mathit{curry}(\mathit{path\text{-}ext}(u, 2 \cdot 2)) = \mathit{lib}@\top@(b@a@\top)@\top$$

In contrast, the ranked path-ext pruning of $\mathit{curry}(u)$ at this leaf yields something quite different:

$$\mathit{path\text{-}ext}(\mathit{curry}(u), 1 \cdot 2 \cdot 1 \cdot 2) = \top@(\top@a@\top)@\top$$

This shows that the framework with general ranked pruning strategies is needed for dealing with the unranked case properly.

The definition of σ-stable queries carries over literally to unranked trees. It follows that a node selection query on unranked trees is stable for an unranked pruning strategy if and only if the corresponding leaf selection query on ranked trees is stable for the corresponding ranked pruning strategy. We can also compare unranked pruning strategies for aggressiveness, literally as we did in the ranked case. It follows as before, that query stability inherits to less aggressive pruning strategies

An annotated unranked tree is a tree in $\mathcal{U}_{\Sigma \cup (\Sigma \times \mathbb{B})}(X)$. It should be noticed that annotated ranked trees correspond to annotated unranked trees in which only leafs are annotated, and vice versa. Pruning functions can be defined on unranked trees literally as in the ranked case. Ranked pruning functions then correspond precisely to unranked pruning functions, whose domains are restricted to leaf-only-annotated trees. Furthermore, we can lift pruning strategies to pruning functions in the unranked case in the same manner as in the ranked case. Based on these correspondences, we can also lift to unranked trees our theorem on learning stable queries from pruned examples without any particular difficulties.

9. Experimental Results

We have implemented our learning algorithm and integrated it into Web and XML information extraction systems. In this section, we present experimental results that illustrate the relevance of schema-guided pruning strategies in practice.

We will start with the presentation of an interactive query induction system and how we simulate it in order to evaluate its performance. We then show how we use our learning

algorithms in the system. Then, we compare our system with existing Web information extraction tools in order to illustrate that our algorithms are competitive. Note that no previous system makes use of the DTD of HTML. We then move to XML information extraction, for which no alternative tool exists for the same tasks to the best of our knowledge.

9.1 Interactive Query Induction

Given a set of XML documents, the goal of the user is to find an unknown target query that selects the correct set of nodes in each of them. In order to do so, the user has to play the role of a teacher who provides the learner with annotations that are consistent with the target query.

9.1.1 Interactive Learning Protocol

The interactive learning protocol follows a classical annotate-learn-select-correct loop. We fix a schema D and as pruning strategy σ either $\mathit{path\text{-}only}_D$ or $\mathit{path\text{-}ext}_D$. At the beginning, the user chooses an XML document from the collection and annotates some nodes as selected or rejected. This yields an unpruned sample S on which the system runs the learning algorithm $\sigma\text{-}\mathit{stable\text{-}learn}_D$. Either a non-stability exception is raised or a σ-stable query is inferred. In case of success, the system presents the answer of the induced query on the current document to the user, possibly via a graphical interface, at least in the case of HTML documents. For other kinds XML documents, there might not exist suitable visualizations, but for example as in the introduction, there are some.

If the user agrees with the query's answer on the current document then he can proceed with inspecting the answer of the same query on another document. Otherwise, he must correct a node that is wrongly selected or rejected and provide a correction by adding a negative or a positive annotation respectively. The learner learns a new query but now from a larger collection of annotated examples, and so on and so forth. The process continues until the user accepts the current query.

9.1.2 Automatic Evaluation

In order to evaluate our learning algorithm $\sigma\text{-}\mathit{stable\text{-}learn}_D$, we will simulate the user in the interactive learning protocol. We assume that the target query is known beforehand, so that we can generate annotations simulating the user's behavior automatically.

Given a totally ordered collection of documents and a pruning strategy σ with schema D, the simulated user behaves as follows in order to find the target query. He creates an empty unpruned sample at the beginning, which is enriched incrementally. He then inspects the first document. If the query does not select any node on this document then the sample remains empty. Otherwise, the first answer node of the document in a breadth-first order is annotated by 1, and the thereby annotated document is added to the sample. The learning algorithm $\sigma\text{-}\mathit{stable\text{-}learn}_D$ is then run with the current sample. It returns a query that is an hypothesis for the target query. If this query returns an incorrect answer set on the current document, then the simulated user corrects the first wrongly selected or rejected node by adding a negative respectively positive annotation, and reruns the learning algorithm with the updated sample. Otherwise, he continues with the next document in the same manner, while always enriching the sample. The simulator has also to decide when the user will stop

the interaction loop. We chose to stop once the current query computes correct answers on 30 consecutive documents or until the whole data collection has been processed. In order to reduce the dependency on the particular total order on the document collection, we will generate 30 different total orders randomly, and report the average results on these 30 repetitions. The quality of the learning algorithm is measured by the following two criteria:

1. the number of annotations to be provided by the simulated user until convergence,
2. the number of XML documents that the simulated user needs to consult until convergence.

These criteria measure the annotation effort of the simulated user until convergence. Its verification effort, on whether the queries proposed by the system are correct, is ignored since considered less relevant.

Compared to a human user, our automatic evaluation procedure with a simulated user is pessimistic in two aspects. First, a human user inspects informative documents eagerly rather than using a random order, and second, he can choose informative annotations and corrections eagerly rather than in breadth first manner.

9.2 Web Information Extraction

We start with a comparison to the Web information extraction tools by Raeymaekers et al. (2008) (see also Raeymaekers, 2008) and Carme et al. (2007). The former is based on learning local tree automata for unranked trees, while the later is based on learning unrestricted deterministic tree automata for ranked trees. No schemas are considered there. Both tools rely on the pruning strategy *path-only* (see the second transformation of Raeymaekers et al., 2008, p. 170), which ignores any schema information. When learning without pruning strategies, both tools yield poor results. We do not present such experiments here but confirm equally poor results for our algorithm. Less aggressive pruning strategies than *path-only* were not tested there. This imposes important restrictions on the class of learnable queries, as we argued here. Furthermore, none of these tools can deal with schema-guided pruning strategies or benefit from the DTD of HTML, in contrast to what we do. Note that we did not try out to work with schemas that got inferred themselves from a collection of XML documents, for instance by the induction algorithm proposed by Bex et al. (2006).

Raeymaekers (2008) provides experimental comparisons to quite some Web information extraction tools based on different methods from machine learning, of which the most efficient are the string-based tools Stalker and Bwi (Muslea et al., 2003). Since his tool is shown competitive with those, and ours is doing equally well for the kind of queries tested there, we skip further comparisons. Unfortunately, however, only few of his datasets are available[4]. As a work around, we rely on the datasets from Carme et al. (2007) for some more challenging cases. All of them are available online[5]. Note that we had to use `tidy`[6] in order to make the HTML documents valid w.r.t. a DTD (XHTML 1.0 Transitional in our case). Indeed, this was necessary for all HTML datasets that we tested with, since none of them was consistent with any existing DTD.

4. This fact got confirmed by Raeymaeker's PhD supervisors by Email contact.
5. See `http://www.grappa.univ-lille3.fr/~carme/WebWiki/DataSets.html`
6. `http://tidy.sourceforge.net`

Table 1: We compare our algorithm *stable-RPNI* with "path-only" pruning function but guided by the schema of HTML with two previous Web information extraction systems. For each tool and dataset, we show the average number of corrections and the average number of documents needed until convergence. For our system, we also show the standard variation over the first 30 experiments. The best result for each target query is highlighted.

Datasets (# of docs.)	Our results		Carme et. al. (07)		Raeymaekers (08)	
	# corr.	# docs.	# corr.	# docs.	# corr.	# docs.
Okra-mails (251)	$2.67_{\pm 0.54}$	$1.77_{\pm 0.42}$	3.48	1.6	**2.0**	?
Bigbook-address (234)	$\mathbf{2.17}_{\pm 0.37}$	$1.17_{\pm 0.37}$	3.02	1	2.3	?
Yahoo (79)	$\mathbf{10.07}_{\pm 3.00}$	$4.27_{\pm 1.34}$	11.36	6.18	–	
Ebay (34)	$\mathbf{1.87}_{\pm 0.67}$	$1.20_{\pm 0.40}$	2.62	1.06	–	
NY-Times (22)	$2.47_{\pm 0.50}$	$1.47_{\pm 0.50}$	**1.44**	1.44	–	
Google (33)	$\mathbf{4.47}_{\pm 0.76}$	$2.57_{\pm 0.56}$	4.78	1.86	–	

The results of our algorithm *p-stable-RPNI*$_{\text{HTML}}$ where $p = p_{\textit{path-only}_{\text{HTML}}}$ are shown in Table 1. They illustrate that our algorithm is fully competitive with both predecessors considered, even though of much larger scope. Indeed, our algorithm often performs best and never needs significantly more annotations that the others. This is probably due to guidance of the pruning strategy by the schema of HTML.

Raeymaekers (2008) did not indicate the number of documents necessary to identify the target queries in the two first datasets, and since his tool is no longer available, we cannot obtain them by other means. The four other datasets were designed by Carme et al. (2007). The Yahoo dataset is the most difficult. This is because the HTML documents in this collection have heterogeneous formats.

9.3 XML Information Extraction

The queries learned for Web information extraction seem to be stable under the *path-only* pruning strategy, since otherwise they could not be learned thereby. In XML information extraction, however, there is a need for more complex queries, where less aggressive pruning strategies must be considered. We next define datasets for such queries and test our learning algorithm on them.

9.3.1 XPath Queries for XML Datasets

An XML dataset is composed of a collection of XML documents, all valid w.r.t. some DTD, and of a special XML document—called *companion*—that enumerates for each document of the collection all nodes that are selected by some fixed XML query (the target). Companions are also used in order to define input samples.

Table 2: Stability of the XML queries used in our experiments and presented in Figure 13 w.r.t. the four pruning strategies on unranked trees.

Query id.	**Stable w.r.t.**			
	path-only	*path-only*$_{\text{XMark}}$	*path-ext*	*path-ext*$_{\text{XMark}}$
XMark-A1	yes	yes	yes	yes
XMark-02	no	yes	yes	yes
XMark-17	no	yes	yes	yes
XMark-21	no	yes	yes	yes
XMark-A8	no	yes	yes	yes
XMark-A6	no	no	no	yes

The datasets we will use in the experiments have been designed upon the XMark benchmark. XMark (Schmidt et al., 2002)[7] is a popular benchmark project in the XML database community. An XMark document stores a set of auctions which contain several data like items, persons, bidders, etc. The main interest of XMark, for our purpose, is that it comes with a rather complex DTD, which defines a set of trees with varied structures[8].

The target queries we use are based on a set of realistic XPath queries that the authors of XMark, as well as Franceschet (2005), have proposed for testing XQuery or XPath processors. We have chosen XPath queries based only on the structure of XMark documents, i.e. queries that do not use tests on data values in their definition. Figure 13 provides the target queries in terms of XPath expressions. All the datasets are available online[9]. Table 2 summarizes the pruning strategies for which these queries are stable.

9.3.2 Experimental Results

For each dataset, we run our learning algorithm with the pruning functions defined above where D is an automaton for the XMark DTD. Either an exception is raised because the target query is not stable for the pruning strategy. Or, it is stable and then we give the number of interactions done by the simulator, the number of pages which have been visited before convergence, and the running time of the algorithm. The results of our experiments are presented in Table 3.

Let us recall that we compared pruning strategies according to their aggressiveness in Figure 7. The results show that the best pruning strategy for learning a query is always the most aggressive one for which the target query is stable. Furthermore, if the maximally aggressive strategies are *path-only*$_D$ and *path-ext* (recall that they can not be compared), then the results show (with the exception of XMark-17) that *path-only*$_D$ is the better. In other words, when a schema is available, one should use the most aggressive pruning strategy guided by the schema.

7. See `http://www.ins.cwi.nl/projects/xmark`
8. See `http://www.ins.cwi.nl/projects/xmark/Assets/auction.dtd`.
9. See `http://grappa.univ-lille3.fr/~champavere/Recherche/datasets/`

XMark-A1 (50 documents):

$/site/closed\_auctions/closed\_auction$
$/annotation/description/text/keyword$

The target query selects the keywords in the description of the closed auctions. It is the most simple of all queries, since the selection of a node only depends on its path from the root. By definition, all pruning functions capture this condition.

XMark-02 (100 documents):

$/site/open\_auctions/open\_auction/bidder[1]/increase$

The target query selects the increase of the first bidder for all open auctions. It is difficult to learn because the selection of a node depends on its position. That is why the query is not *path-only*-stable.

XMark-17 (100 documents):

$/site/people/person[\texttt{not}(homepage)]/name$

The target query selects the name of the persons that do not have a homepage. The difficulty here is to infer the previous negative condition.

XMark-21 (50 documents):

$/site/open\_auctions/open\_auction[count(bidder) \geq 3]/itemref$

The target query selects the item reference of all open auctions whose numbers of bidders is greater than three. It is hard to learn because the selection of one node depends on information on its siblings. This explains why it is not stable by *path-only*-pruning.

XMark-A8 (100 documents):

$/site/people/person[address\ \texttt{and}\ (phone\ \texttt{or}\ homepage)$
$\texttt{and}\ (creditcard\ \texttt{or}\ profile)]/name$

The target query selects the name of the persons who have filled in several information. The learning algorithm must infer a conjunction of disjunctions, which is a hard task.

XMark-A6 (250 documents):

$/site/people/person[profile/gender\ \texttt{and}\ profile/age]/name$

The target query selects the name of persons who have filled in both their gender and their age in their profile. This is a difficult query because the selected nodes depend on two children of their siblings. Only *path-ext*$_D$-pruning is able to capture this condition.

Figure 13: Description of XML queries used in our experiments.

Table 3: Experiments on XML data: average number of corrections and documents, and the total time needed to infer queries with respect to pruning strategies.

XMark-A1	avg. # corrections	avg. # documents	total time (s)
path-only	**1.40** ±0.49	1.40 ±0.49	0.16 ±0.15
path-only$_D$	4.60 ±0.99	3.10 ±0.75	1.17 ±0.70
path-ext	6.03 ±1.64	4.10 ±1.30	4.75 ±2.66
path-ext$_D$	8.87 ±2.08	5.80 ±1.38	14.77 ±5.66

XMark-02			
path-only	exception 'unstable query'		
path-only$_D$	**4.43** ±1.12	3.30 ±0.69	0.54 ±0.35
path-ext	13.33 ±7.74	7.43 ±3.22	9.17 ±8.82
path-ext$_D$	18.10 ±6.38	14.07 ±4.09	20.62 ±11.48

XMark-17			
path-only	exception 'unstable query'		
path-only$_D$	13.57 ±1.54	4.77 ±1.31	1.09 ±0.36
path-ext	**4.90** ±0.98	2.93 ±0.81	0.42 ±0.21
path-ext$_D$	20.43 ±2.70	6.90 ±1.56	6.32 ±2.04

XMark-21			
path-only	exception 'unstable query'		
path-only$_D$	**12.00** ±2.78	7.80 ±1.66	4.17 ±1.74
path-ext	22.97 ±4.69	16.03 ±3.04	47.65 ±21.63
path-ext$_D$	40.47 ±7.08	31.50 ±5.52	83.39 ±29.74

XMark-A8			
path-only	exception 'unstable query'		
path-only$_D$	**11.93** ±6.62	5.27 ±1.57	5.95 ±11.86
path-ext	124.87 ±59.34	35.87 ±17.31	872.29 ±1240.39
path-ext$_D$	32.10 ±18.04	14.30 ±6.75	82.79 ±169.44

XMark-A6			
path-only	exception 'unstable query'		
path-only$_D$			
path-ext			
path-ext$_D$	**16.03** ±2.74	9.30 ±2.21	12.83 ±5.60

10. Conclusion and Future Work

We distinguished classes of stable queries for schema-guided pruning strategies, and proposed new learning algorithms for regular stable queries. Experimental evidence shows that stability is the essential notion for understanding the difficulty of particular queries, in that

queries are easier to learn if they remain stable under more aggressive pruning strategies. Furthermore, schema guidance is useful for defining relevant pruning strategies.

Which classes of XPath queries can be learned with what kind of pruning strategies remains to be discussed. That is the question, which classes of XPath queries are stable for which pruning strategies. Simple XPath queries with forward child and descendant axis only, such as `//a/b//c/d`, are stable under the *path-only* pruning strategy. When adding filters that use the child axis once such as `//a[./b]/c[./d]/e` they remain stable for *path-only*$_D$. When permitting a second child axis in filters, such as in `//a[./b/f]/c[./d/g]/e`, we obtain a class of XPath queries that is stable under the *path-ext*$_D$ strategy. Pruning may become useless for queries with descendant axis in filters, such as `/a[.//b]`, where node selection depends on arbitrary b-descendants. Whether this happens also depends on the schema. Similar problems are raised when using other recursive axis in filters. When using the "following" axis on the main path, such as in `//a/following::b`, we can use yet another pruning function that keeps all nodes following a selected node in document order (or only the next b-node following a selected node). Recursive backward axis, such as in `//a/preceding::*//c` may also quickly lead to non-stability. For more general classes of XPath queries, an interesting problem that we leave open might be to learn suitable pruning strategies. A user could help by annotating nodes that are relevant for selecting others.

Another question is how to deal with XPath queries with tests on data values, such as `//book[./author="Knuth"]`. A practical approach could be to enrich the interactive setting by allowing the user to specify data values of interest. Another approach should be to combine our approach with statistical learning methods dealing with data values. Queries, in which equality of data values can be tested (joins), may be non-regular and thus raise more principal difficulties to learnability. More generally, an interesting problem is whether classes of path queries for graph databases can be learned and what a pruning strategy could be in this context.

Also, in the future, query induction with schema-guided pruning strategies should be extended to n-ary regular queries (Lemay et al., 2006). This new framework for pruning strategies provided in this paper should be sufficiently general, so that one can define appropriate pruning strategies in the n-ary case (in contrast to previous settings). Other interesting directions would be to study OXPath queries (Sellers et al., 2011b) or tree-to-tree transformations (Lemay et al., 2010).

Acknowledgements. We thank the anonymous reviewers for their excellent work. They read all proofs in great detail while proposing many improvements. In particular, one of them spotted an error in a previous version of Proposition 5 which lead us to the important distinction between pruning strategies and pruning functions. We would also like to thank F. Gire and J.-M. Champarnaud for their valuable comments when reviewing J. Champavère's PhD thesis which preceded the present article. This work was partially supported by the French National Research Agency (ANR) through project Codex of the program ANR-08-DEFIS-004 (2009-2012).

References

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

Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *28th International Conference on Very Large Data Bases (VLDB)*, pages 119–128, 2001.

Geert J. Bex, Frank Neven, Thomas Schwentick, and Karl Tuyls. Inference of concise DTDs from XML data. In *32nd International Conference on Very Large Data Bases (VLDB)*, pages 115–126, 2006.

Julien Carme, Joachim Niehren, and Marc Tommasi. Querying unranked trees with stepwise tree automata. In *19th International Conference on Rewriting Techniques and Applications (RTA)*, pages 105–118, 2004.

Julien Carme, Michal Ceresna, Oliver Frölich, Georg Gottlob, Tamir Hassan, Marcus Herzog, Wolfgang Holzinger, and Bernhard Krüpl. The Lixto project: Exploring new frontiers of web data extraction. In *23rd International Information Systems Conference (BNCOD)*, pages 1–15, 2006.

Julien Carme, Michal Ceresna, and Max Goebel. Query-Based learning of XPath expressions. In *8th International Colloquium on Grammatical Inference (ICGI)*, pages 342–343, 2006.

Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1):33–67, 2007.

Jérôme Champavère. *Induction de requêtes guidée par schémas*. PhD thesis, Université des Sciences et Technologies de Lille 1, 2010.

Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Schema-Guided Induction of Monadic Queries. In *9th International Colloquium on Grammatical Inference (ICGI)*, pages 15–28, 2008.

Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Efficient inclusion checking for deterministic tree automata and XML schemas. *Information and Computation*, 207(11):1181–1208, 2009.

William W. Cohen, Matthew Hurst, and Lee S. Jensen. A flexible learning system for wrapping tables and lists in HTML documents. In *11th International Conference on World Wide Web (WWW)*, pages 232–241, 2002.

Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Electronic book available at `http://tata.gforge.inria.fr/`. Revised 2007.

François Coste, Daniel Fredouille, Christopher Kermorvant, and Colin de la Higuera. Introducing domain and typing bias in automata inference. In *7th International Colloquium on Grammatical Inference (ICGI)*, pages 115–126, 2004.

Massimo Franceschet. XPathMark: An XPath benchmark for the XMark generated data. In *3rd International Conference on Database and XML (XSym)*, pages 129–143, 2005.

Dayne Freitag and Andrew K. McCallum. Information extraction with HMMs and shrinkage. In *AAAI Workshop on Machine Learning for Information Extraction*, pages 31–36, 1999.

Pedro García and Jose Oncina. Inference of recognizable tree sets. Technical Report DSIC-II/47/93, Universidad de Alicante, 1993.

Rémi Gilleron, Florent Jousse, Isabelle Tellier, and Marc Tommasi. XML document transformation with conditional random fields. In *5th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, pages 525–539, 2006.

Rémi Gilleron, Patrick Marty, Marc Tommasi, and Fabien Torre. Interactive tuples extraction from Semi-Structured data. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 997–1004, 2006.

Georg Gottlob and Christoph Koch. Monadic queries over Tree-Structured data. In *17th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 189–202, 2002.

Georg Gottlob, Erich Grädel, and Helmut Veith. Datalog LITE: a Deductive Query Language with Linear Time Model Checking. *ACM Transactions on Computational Logics*, 3(1): 42–79, 2002.

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

Aurélien Lemay, Joachim Niehren, and Rémi Gilleron. Learning n-ary node selecting tree transducers from completely annotated examples. In *8th International Colloquium on Grammatical Inference (ICGI)*, pages 253–267, 2006.

Aurélien Lemay, Sebastian Maneth, and Joachim Niehren. A learning algorithm for Top-Down XML transformations. In *29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 285–296, 2010.

Wolfgang May. Information extraction and integration with Florid: The Mondial case study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999.

Michel Minoux. LTUR: a Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and Computer Implementation. *Information Processing Letters*, 29(1):1–12, 1988.

Ion Muslea, Steve Minton, and Craig Knoblock. Active learning with strong and weak views: a case study on wrapper induction. In *18th International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 415–420, 2003.

Jose Oncina and Miguel A. Varó. Using domain information during the learning of a subsequential transducer. In *3rd International Colloquium on Grammatical Inference (ICGI)*, pages 301–312, 1996.

David Pinto, Andrew McCallum, Xing Lee, and W .Bruce Croft. Table extraction using conditional random fields. In *26th ACM SIGIR*, pages 235–242, 2003.

Stefan Raeymaekers. *Information extraction from Web pages based on tree automata induction.* PhD thesis, Katholieke Universiteit Leuven, 2008.

Stefan Raeymaekers, Maurice Bruynooghe, and Jan Van den Bussche. Learning (k, l)-contextual tree languages for information extraction from web pages. *Machine Learning*, 71(2-3):155–183, 2008.

Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. XMark: A benchmark for XML data management. In *28th International Conference on Very Large Data Bases (VLDB)*, pages 974–985, 2002.

Andrew J. Sellers, Tim Furche, Georg Gottlob, Giovanni Grasso, and Christian Schallhart. Taking the OXPath down the deep web. In *14th International Conference on Extending Database Technology (EDBT)*, pages 542–545, 2011.

Andrew J. Sellers, Tim Furche, Georg Gottlob, Giovanni Grasso, and Christian Schallhart. OXPath: little language, little memory, great value. In *WWW (Companion Volume)*, pages 261–264, 2011.

Slawomir Staworko and Piotr Wieczorek. Learning Twig and Path Queries. In *15th International Conference on Database Theory (ICDT)*, 2012.

Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. 2D conditional random fields for web information extraction. In *22nd International Conference on Machine Learning (ICML)*, pages 1044–1051, 2005.

Appendix A. Remaining Proofs

Lemma 13 (Open Part) *For any tree automaton A recognizing D-pruned annotated trees, the language of unpruned trees $\mathcal{L}_{\mathcal{Q}_A}$ of the query $\mathcal{Q}_A$ with schema D is regular.*

Proof We have to construct a tree automaton A' recognizing $\mathcal{L}_{\mathcal{Q}_A}$. It must check for a tree $t * \beta$ whether all 1-annotations of β are justified by some 1-annotation of a tree $t' * \beta' \in \mathcal{L}(A)$ where $t \in \mathit{compl}_D(t')$, and that no 0-annotation of β is in conflict with some 1-annotation of a tree $t'' * \beta'' \in \mathcal{L}(A)$ with $t \in \mathit{compl}_D(t'')$. We first construct a deterministic automaton A'' that recognizes the language:

$$\mathcal{L}(A'') = \{t * \beta \mid t \in \mathcal{L}(D),\ t \in \mathit{compl}_D(t'),\ t' * \beta \in \mathcal{L}(A)\}$$

This is straightforward but requires determinization. Given a tree t let γ_t be the function that maps nodes ν of t to the state of A' that contains all states r such that there exists β for which $t[\nu \mapsto r] * \beta$ is recognized by A''. We construct an automaton A''' that recognizes the language:

$$\mathcal{L}(A''') = \{t * \beta * \gamma_t \mid t \in \mathcal{L}(D),\ \beta \text{ is } Q\text{-consistent}\}$$

Once this will be done, we can define A' by projection $A' = \Pi_{\Sigma \cup (\Sigma \times \mathbb{B})}(A''')$. For trees $t * \beta * \gamma$, it can be tested by an automaton whether $\gamma = \gamma_t$ and also whether $t \in \mathcal{L}(D)$. What remains to construct is an automaton that tests for trees $t * \beta * \gamma$ whether β is Q-consistent. This can be done by an automaton B that we define next. The states of B are the subsets of states of A''. For $f \in \Sigma^{(n)}$, a set of symbols $F \subseteq \{f, (f,0), (f,1)\}$, and states $R_1, \ldots, R_n$ of A'' we define the following set of states:

$$F(R_1, \ldots, R_n) = \{r \mid\ \tilde{f} \in F,\ \tilde{f}(r_1, \ldots, r_n) \to r \text{ in } A'',\ r_i \in R_i\}$$

These are the states that A'' can reach from some tuple of states in $R_1 \times \ldots \times R_n$ with a symbol from F. The rules of B are inferred as follows where $f \in \Sigma^{(n)}$.

$$\frac{R \cap \{(f,1)\}(R_1, \ldots, R_n) \neq \emptyset \qquad R' = F(R_1, \ldots, R_n) \qquad F = \{(f,0), (f,1), f\}}{((f,1), R)(R_1, \ldots, R_n) \to R' \text{ in } B}$$

$$\frac{R \cap \{(f,1)\}(R_1, \ldots, R_n) = \emptyset \qquad R' = F(R_1, \ldots, R_n) \qquad F = \{(f,0), (f,1), f\}}{((f,0), R)(R_1, \ldots, R_n) \to R' \text{ in } B}$$

$$\frac{R' = F(R_1, \ldots, R_n) \qquad F = \{(f,0), (f,1), f\}}{(f, R)(R_1, \ldots, R_n) \to R' \text{ in } B}$$

The final states of B are those states R that contain some final state of B. If B goes into a state R on a tree $t * \beta * \gamma$ then all states in R can be reached on some tree $t * \beta' \in \mathcal{L}(A'')$ and all annotations of β are compatible with γ. If $\gamma = \gamma_t$, then β is Q-consistent. ■

Theorem 19 *Let D be a deterministic tree automaton with signature Σ, P a deterministic tree automaton with signature $(\Sigma \cup (\Sigma \times \mathbb{B})) \times \{y, n\}$, $p = \wp_{P,D}$ a pruning function, and Q a query with domain D. Given a tree automaton A with $\mathcal{L}(A) = \mathcal{L}_Q$ we can decide in time $O(|A|^2\ |D|\ |P|)$ whether Q is p-stable.*

Proof We write $t_1 \circledast \ldots \circledast t_n$ for the overlay of trees t_1, ..., t_n over possibly different ranked signatures, where missing nodes are filled up with a fresh $\perp$-symbol. We consider the following tree language:

$$\left\{ \begin{array}{c} t * \beta \circledast t' * \beta' \circledast t_1 * \beta_1 \;\mid\; t' * \beta' = p(t * \beta),\ t_1 \in \mathit{compl}_D(t'), \\ \beta \text{ is } Q\text{-consistent for } t,\ \beta_1 \text{ is } Q\text{-consistent for } t_1, \\ \exists \nu \in \mathit{nodes}(t').\ \beta(\nu) = 1 \wedge \beta_1(\nu) = 0 \end{array} \right\}$$

By construction, this language is empty and if only if Q is p-stable. This can be decided in linear time in the size of an automaton recognizing the above language:

- By running a single copy of D jointly on $t' \circledast t_1$ – as explained in more details in the proof of Theorem 22 – one can check whether $t_1 \in \mathit{compl}_D(t')$.
- By running A on the first component in parallel, we can check whether β is a Q-consistent annotation of t, and thus whether $t \in \mathcal{L}(D)$.
- By running $\Pi_{\Sigma \cup (\Sigma \times \mathbb{B})}(P)$ on $t * \beta$, we can check that the second component $t' * \beta'$ is the p-pruning of the first component $t * \beta$.
- By running another copy of A on the third component in parallel, we can check whether β_1 is a Q-consistent annotation of t_1, and thus whether $t_1 \in \mathcal{L}(D)$.
- Testing whether $\exists \nu \in \mathit{nodes}(t').\ \beta(\nu) = 1 \wedge \beta_1(\nu) = 0$ can be done with a 2-state control.

In summary, we have to run the following automata in parallel on different components: twice A, once D and once P, so the overall size of the automaton is in $O(|A|^2\ |D|\ |P|)$. ■