
Towards Schema-Guided XML Query Induction

Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, Joachim Niehren
INRIA Futurs, Lille University, France

Abstract

XML query induction is a key task in Web information extraction. Recent approaches based on grammatical inference represent node selection queries in XML trees by deterministic tree automata. In this paper, we show how to guide RPNI-based learning algorithms by XML schemas which we can infer in a preprocessing step. We hope that schema guidance will help to improve heuristics that are essential for query learning algorithms.

1. Introduction

Web information extraction aims at distinguishing informative parts of HTML or XML documents. The notion of informativeness depends on the application and is subject to definition. Queries for informative information (often called wrappers) need either to be programmed in some XML query language (Gottlob & Koch, 2004) or inferred by XML query induction from a set of user-annotated examples (Muslea et al., 2002; Chidlovskii, 2001; Cohen et al., 2003).

Induction algorithms for node selection queries in XML trees are particularly relevant for information extraction from well-structured Web pages that layout some kind of database information (Carme et al., 2006; Raeymaekers et al., 2005). Most typically, one might want to extract product-price pairs in some shopping offer. The tree structure of such kinds of documents is usually sufficient for locating the required information, so that texts can be ignored.

Queries learned by the above induction algorithms typically fail to satisfy the schema of XML documents of the target application, in that they are able to select information from Web pages that should not exist. This is not a problem as such, as long as only correct nodes are selected in all documents of the con-

sidered application. Schema violations, however, are often raised by wrong generalizations during the learning process, which may lead to wrong node selection on so far unseen documents. Wrong generalizations need to be avoided as far as possible. This is usually done by numerous heuristics (rarely described in research papers) on which the success of the learning process depends.

In this paper, we investigate schema-guided query learning algorithms on basis of RPNI variants for trees (Carme et al., 2007; Lemay et al., 2006; Oncina & García, 1993). We consider Boolean and monadic queries in trees that are represented by deterministic tree automata. We represent schemas by XML document type definitions (DTDs), whose regular expressions are one-unambiguous (or deterministic), so that the corresponding Glushkov automata are deterministic (Brüggemann-Klein & Wood, 1998), or more generally, by deterministic tree automata. The DTD of HTML is the most typical example. Or else, we infer reasonably concise schemas in terms of 2-testable tree automata as proposed by (Bex et al., 2006).

A schema-guided variant of RPNI for finite word automata has been presented in (Coste et al., 2004). It refutes state merges if the language of the current automaton is no more included in the language defined by the schema. In a first step, we improve this algorithm such that it avoids computing the complement schema and extend it to tree automata for ranked trees (nodes have a fixed number of children) and unranked trees (nodes may have an unbounded number of children). Note that if the schema is represented by a tree automaton D , the size of the complement schema $|D^c|$ includes a factor $|\text{states}(D)|^k$ where k is the maximal arity of symbols and testing inclusion between the hypothesis language and the language defined by the schema also includes this factor. Thus we will rely on a recent inclusion test from (Champavère et al., 2007) which avoids computing D^c and which has time complexity in $O(|A| \times |D|)$ where $|A|$ is the size of the hypothesis automaton and $|D|$ is the size of the schema. This result applies for ranked trees whatever

is the maximal arity of symbols and for unranked trees.

In a second step, we present a schema-guided induction algorithm for monadic node selection queries. These select nodes in trees satisfying the schema, and return the empty set of nodes for all others. We show how to integrate schema inclusion tests into the RPNI variant of (Carme et al., 2007). Finally, we extend this algorithm for learning from partial annotations (only a subset of nodes to be selected are annotated by the user) defining an improved schema-sensitive pruning heuristic. We hope for improvements of the learning quality in practice, even though experimental results are not yet available.

2. Schema-Guided Language Induction

We present an RPNI variant for schema-guided induction of binary tree languages. The target language must be included in the schema’s language. Trees that are inconsistent with the schema are negative examples. We use deterministic tree automata as schemas and for representing the target language. Our learning algorithm will test language inclusion for such automata in an incremental manner.

In this section, we consider binary trees build from some signature Σ with a single binary function symbol $@$ and finitely many constants. We denote by T_Σ the set of trees that can be build over this signature. A *binary tree* $t \in T_\Sigma$ is a constant $a \in \Sigma$ or a triple $@(t_1, t_2)$ that we write $t_1@t_2$ for some $t_1, t_2 \in T_\Sigma$. A (bottom-up) *tree automaton* A over Σ is a tuple $(\Sigma, \text{states}(A), \text{final}(A), \text{rules}(A))$. It contains a finite set $\text{states}(A)$ of states, a set $\text{final}(A) \subseteq \text{states}(A)$ of final states, and a finite set $\text{rules}(A)$ of rules of one of the following forms: $a \rightarrow p$, or $p_1@p_2 \rightarrow p$, or $p_1 \xrightarrow{\epsilon} p_2$ where $a \in \Sigma$ and $p, p_1, p_2 \in \text{states}(A)$. We call an automaton A (bottom-up) *deterministic* if it does not contain epsilon rules and if no two of its rules have the same left-hand side. The size $|A|$ of a tree automaton A is the sum of the cardinalities of its state and rule sets. This measure is independent of the number of symbols in Σ , so that only the symbols that occur in rules of A are counted.

A tree automaton A over Σ *evaluates* binary trees $t \in T_\Sigma$ to sets of states by the function $\text{eval}_A : T_\Sigma \rightarrow 2^{\text{states}(A)}$. The evaluation is inductively defined by $\text{eval}_A(a) = \{p \mid a \rightarrow p' \xrightarrow{\epsilon^*} p \in \text{rules}(A)\}$ and $\text{eval}_A(t_1@t_2) = \{p \mid p_1 \in \text{eval}_A(t_1), p_2 \in \text{eval}_A(t_2), p_1@p_2 \rightarrow p' \xrightarrow{\epsilon^*} p \in \text{rules}(A)\}$. A tree t is *recognized* by A if $p \in \text{final}(A)$ for some $p \in \text{eval}_A(t)$. The *language* of A , denoted $L(A)$, is the set of trees recognized by A . We call a state p of A *accessible* if

there exists a binary tree t such that $p \in \text{eval}_A(t)$, and *productive* if it is used for evaluating some tree recognized by A . We call a tree automaton *productive* if all of its states are.

2.1. Inclusion Test

Given a possibly non-deterministic tree automaton A and a deterministic tree automaton D over the same signature, we test for language inclusion $L(A) \subseteq L(D)$ without computing the complement D^c . Instead, the idea is to compute the accessible states of the product automaton $A \times D$ and to test for the following three failure conditions:

failure₁: there exists a rule $a \rightarrow p \in \text{rules}(A)$ but no state $q \in \text{states}(D)$ such that $a \rightarrow q \in \text{rules}(D)$.

failure₂: there exist accessible states (p_1, q_1) and (p_2, q_2) of $A \times D$ and a rule $p_1@p_2 \rightarrow p \in \text{rules}(A)$ but no state $q \in \text{states}(A)$ such that $q_1@q_2 \rightarrow q \in \text{rules}(D)$.

failure₃: there exists an accessible state (p, q) of $A \times D$ such that $p \in \text{final}(A)$ but $q \notin \text{final}(D)$.

These three conditions characterize inclusion under the assumption that A is productive which we can assume without loss of generality.

Proposition 1 *If A is productive and D deterministic then $L(A) \not\subseteq L(D)$ if and only if **failure₁** or **failure₂** or **failure₃**.*

Computing the set of accessible states of $A \times D$ can be easily done in time $O(|A| * |D|)$. It is also straightforward to test **failure₁** and **failure₃**. The nontrivial question is how to test **failure₂** efficiently. Naive approaches will lead to $O(|A| * |\text{states}(D)|^2)$ algorithms, which is as bad as computing D^c directly. A better solution is given in (Champavère et al., 2007):

Proposition 2 *Conditions **failure₁**, **failure₂**, and **failure₃** can be tested in time $O(|A| * |D|)$.*

Note that the complexity bound does not depend on the size of the common signature of A and D . The algorithm is incremental with respect to the addition of epsilon rules to automaton A . We do not consider this in the complexity analysis though.

2.2. Schema-Guided RPNI

A *sample* S for a target language is a set of positive examples (trees in the target language) together with a set of negative examples (trees in the complement of

the target language). The schema-guided RPNI algorithm inputs a schema D that is a deterministic tree automaton over Σ and a sample S of trees in $L(D)$ for some target language. An hypothesis language is represented by a deterministic tree automaton A . In order to test language inclusion $L(A) \subseteq L(D)$ in an incremental fashion, RPNI maintains an automaton B with epsilon rules such that $L(B) = L(A)$ and tests for $L(B) \subseteq L(D)$. Whenever two states p_1 and p_2 of A are merged, new epsilon rules $p_1 \xrightarrow{\epsilon} p_2$ and $p_2 \xrightarrow{\epsilon} p_1$ are added to B and inclusion in D is reinspected incrementally. RPNI will use the usual deterministic state merging for A in addition to the following procedures:

- `init(B, D)` initializes the data structures for the inclusion test $L(B) \subseteq L(D)$ in time $O(|B| * |D|)$.
- `merge(B, p_1, p_2, D)` adds epsilon rules $p_1 \xrightarrow{\epsilon} p_2$ and $p_2 \xrightarrow{\epsilon} p_1$ to `rules(B)` and updates the data structures for the inclusion test incrementally. In the worst case, it runs in time $O(|B| * |D|)$ but it may be much more efficient in practice due to incrementality.
- `inclusion_test(B, D)` tests whether $L(B) \subseteq L(D)$. This procedure answers in constant time as the whole computation is actually performed by the `merge` and `init` procedures.

The schema-guided RPNI variant first builds an initial automaton A from the positive examples in S as usual¹. It then creates B as a copy of A and initializes the data structures for inclusion checking by calling `init(B, D)`. Then, for all $p \in \text{states}(A)$, RPNI calls the deterministic merge procedure as usual on other states $p' \in \text{states}(A)$. For testing consistency with the schema, it calls `merge(B, p, p', D)`. It accepts the merge if `inclusion_test(B, D)` returns true; otherwise, it backtracks the merge operation and the data structures for inclusion checking, before continuing.

Proposition 3 *For a schema D and a sample S of positive examples, the run time of the schema-guided RPNI algorithm is $O(|S|^3 * |D|)$.*

The `init` procedure has complexity in $O(|S| * |D|)$ where $|S|$ is the total number of nodes of trees in S . The `merge` procedure is called for all merging attempt of A . There are at most $|S|^2$ merges. The complexity for the inclusion test is $O(|S| * |D|)$. Thus the overall

¹The initial automaton is build as in (Oncina & García, 1993) by building the largest deterministic tree automaton that strictly recognizes S . This is related to the prefix tree automaton of RPNI in words (Oncina & Garcia, 1992).

complexity of RPNI with this inclusion test is $O(|S|^3 * |D|)$.

Note that (Coste et al., 2004) assume that an automaton D^c representing the complement of the schema D is given as input, in contrast to what we do. Note also that our inclusion procedure could be used quite straightforwardly for saturation-based algorithms such as DeLeTe2 (Denis et al., 2004).

2.3. Schema-Guided Learning Model

Using schema guidance for learning regular tree languages leads to verify that the new strategy still conforms the model from polynomial time and data of (Gold, 1967; de la Higuera, 1997). Before, let us recall that a tree automaton A (and its language) is said to be *compatible with* a sample S if all positive examples of S are recognized by A whereas all negative examples are not. Conversely we say that S is *consistent with* A . A schema-guided learning model is defined below.

Definition 1 Tree languages represented by a class of tree automata \mathcal{A} are *identifiable from polynomial time and data w.r.t. schemas represented by a class of tree automata \mathcal{D}* if there exist two polynomials p_1 and p_2 , and an algorithm *learner* such that:

(i) for each tree automaton D of \mathcal{D} for the schema and each sample S of trees in $L(D)$, there exists an automaton $A \in \mathcal{A}$ which is compatible with S such that $L(A) \subseteq L(D)$, $\text{learner}(S) = A$ and A can be computed in time $O(p_1(|S|, |D|))$;

(ii) for each tree automaton D of \mathcal{D} and for each automaton A of \mathcal{A} such that $L(A) \subseteq L(D)$, there exists a (characteristic) sample S_A of trees in $L(D)$ with cardinality less than $p_2(|A|)$ such that, having a sample $S \supseteq S_A$ of trees in $L(D)$ as input, *learner* returns a tree automaton equivalent to A .

We now show that regular tree languages represented by deterministic tree automata are identifiable from polynomial time and data w.r.t. schemas represented by deterministic tree automata. The algorithm *learner* is defined to be the schema-guided RPNI previously defined. The polynomial p_1 is defined by $p_1(|S|, |D|) = |S|^3 * |D|$. Algorithm *learner* returns an automaton compatible with S because it is a property of RPNI. By definition, this one always preserves inclusion of the hypothesis language in $L(D)$. The complexity result has been proved above.

Recall that regular tree languages represented by deterministic tree automata are identifiable from polynomial time and data with algorithm RPNI. Thus, there

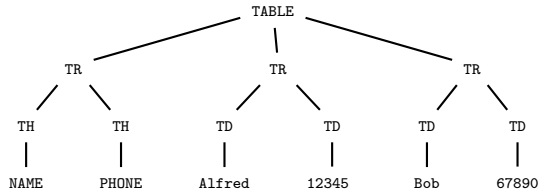


Figure 1. The tree `phone-list` represents an HTML table containing list of phone numbers.

exists, for each deterministic tree automaton A , a characteristic sample, that we denote by R_A . The cardinality of R_A is less than $p_2(|A|)$. With a sample $S \supseteq R_A$ as input, RPNI returns a tree automaton equivalent to A . Let R_A^+ be the set of positive examples and R_A^- the set of negative examples of R_A . We define $S_A^+ = R_A^+$, $S_A^- = R_A^- \cap L(D)$, and the characteristic sample S_A of A in the model w.r.t. schemas to be the sample whose positive examples are in the set S_A^+ and negative examples in the set S_A^- . The cardinality of S_A is less than $p_2(|A|)$. Now, let us consider a sample $S \supseteq S_A$ of trees in $L(D)$. The algorithm *learner* always considers hypotheses which are compatible with S because it is derived from RPNI. Moreover, *learner* always preserves inclusion of the hypothesis language in $L(D)$. Therefore, each hypothesis language is always compatible with negative examples in the complement of $L(D)$, thus with examples in R_A^- which are not in $L(D)$. Algorithm *learner* with S as input outputs the same automaton than RPNI with $S \cup (T_\Sigma - L(D))$ as input. Since $R_A \subseteq (S \cup (T_\Sigma - L(D)))$ and using a property of RPNI, we obtain that *learner* returns a tree automaton equivalent to A .

3. XML Schema Induction

3.1. DTDs and Stepwise Tree Automata

XML trees are finite sibling-order unranked trees built from an unranked signature Σ of node labels. An *unranked tree* t is a pair $a(t_1, \dots, t_n)$ consisting of a label $a \in \Sigma$ and a possibly empty sequence of unranked trees t_1, \dots, t_n over Σ . Let us consider for instance documents that are HTML tables that contain a list of persons with their phone number. The tree `phone-list` is an example of such a document described in Figure 1. It is represented by an unranked tree because the number of children under the symbol `TABLE` is unbounded.

Unrankedness is usually eliminated by encoding into binary trees. We will work with the Curried encoding from stepwise tree automata (Carme et al., 2004) which encodes unranked trees over Σ into binary trees over $\Sigma \cup \{\@\}$. Stepwise tree automata on unranked

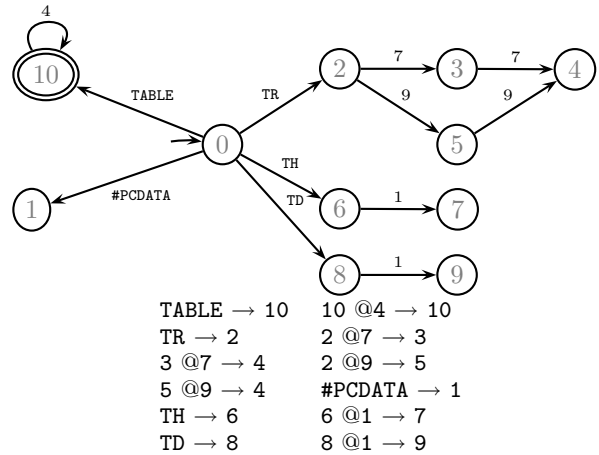


Figure 2. A stepwise tree automaton corresponding to `schema-phone`, graphically and by rules.

trees can be identified with binary tree automata on Curried encodings. They are advantageous for minimization (Martens & Niehren, 2007) and query induction (Carme et al., 2007; Lemay et al., 2006). For instance a stepwise tree automaton for unranked trees corresponding to HTML tables containing lists of phone numbers is given in Figure 2.

A stepwise automaton can be understood as a regular string automata reading at each node of the input tree the string formed by the label of the node and the concatenation of the states of its children (if any). For instance, let us consider a run of the automaton of Figure 2 on the tree `phone-list`. The node containing *Alfred* will be put in the state 1 (we read `#PCDATA`), the TD node above will be put in the state 9 (we read `TD.1`), the TR node above will be put in the state 4 (we read `TR.9.9`), and the root node will be put in the state 10 by reading `TABLE.4.4.4`. As this state is final, this tree is recognized.

Alternatively, the stepwise can be seen as a traditional binary tree automaton operating on the Curried encoding of the tree. For example, the subtree of `phone-list` under the TR node above *Alfred* can be represented by `@(@ (TR, @(TD, Alfred)), @(TD, 12345))` using Curried encoding, and the above stepwise—viewed as a classical tree automaton (rules are on bottom of Figure 2)—will associate the state 4 to the root of this tree as above. See (Carme et al., 2004) for more details.

Various XML schema languages have been proposed for defining regular types of XML trees. In this paper, we will be mostly interested in DTDs and their relation to deterministic tree automata. The alternative schema formalisms XML Schema and Relax NG are

closely related to tree automata too.

A possible DTD for HTML tables containing lists of phone numbers is given in Figure 3. DTDs are defined using one-unambiguous regular expressions as recommended by the W3C (Brüggemann-Klein & Wood, 1998). A regular expression is one-unambiguous iff its Glushkov automaton is a deterministic finite automaton. DTDs whose rules use deterministic finite automata can be compiled into deterministic stepwise tree automata (Martens & Niehren, 2007). The overall translation is in quadratic time. It can be brought down to linear time when compiling into deterministic factorized stepwise tree automata instead and complexity results for inclusion can be extended to these automata (Champavère et al., 2007). This way, testing inclusion between the language $L(A)$ recognized by a tree automaton A in the language $L(D)$ defined by a DTD D can be done in time $O(|A| \times |D|)$.

3.2. Inference of DTDs

HTML documents satisfy the DTD of HTML that the W3C provides². This DTD, however, does not specify more concrete schemas of particular HTML Web site. Even worse, such schemas are hardly available in any explicit form at all. For instance a schema for HTML tables that contain a list of persons with their phone number could be the part of the HTML DTD that describes tables. Another schema is the DTD `schema-phone` of Figure 3 which is adapted to this particular kind of tables: they have only two columns that contain either TH nodes or TD nodes. The second schema is of course better and should allow to improve the learning process. We have seen that DTDs can also be represented by stepwise automata, and the one in Figure 2 corresponds to the DTD in Figure 3. Unfortunately, the latter is probably not directly available, and thus it should be inferred.

```
<!ELEMENT TABLE (TR*) >
<!ELEMENT TR (TH, TH|TD, TD)>
<!ELEMENT TH (#PCDATA)>
<!ELEMENT TD (#PCDATA)>
```

Figure 3. A DTD for `schema-phone`.

An inference algorithm from positive examples for DTDs was proposed by (Bex et al., 2006). It uses 2-testable word languages and we rely on this algorithm. We illustrate the inference algorithm in the sequel of the section. It should be noted that, in our framework, readable DTDs are not required because they are only used in inclusion tests to improve the learning process.

²See for instance <http://www.w3.org/TR/html401/>.

Given a set of trees, the inference algorithm makes a depth-first scan of each tree in order to collect the following information: for each label l (XML or HTML tag), it builds the set of words consisting in the sequence of labels of the children of nodes of label l . For instance, in the `phone-list` tree, under the label TR, sequences of labels TH TH and TD TD are observed. From those sets of words, it generalizes the language under each label using the classical 2-testable language induction algorithm of (Garcia & Vidal, 1990). In this example, the obtained language for TR is TH.TH*|TD.TD*. The resulting language then naturally translates into a stepwise tree automaton. As an example, the inference algorithm would output the DTD given in Figure 4 with the `phone-list` tree as input.

```
<!ELEMENT TABLE (TR, TR*)>
<!ELEMENT TR (TH, TH*|TD, TD*)>
<!ELEMENT TH (#PCDATA)>
<!ELEMENT TD (#PCDATA)>
```

Figure 4. An inferred DTD for `schema-phone`.

4. Schema-Guided Query Induction

We consider the RPNI algorithm from (Carme et al., 2007) for learning monadic node selection queries in unranked trees and extend it by schema guidance.

We first recall how to define monadic queries by tree automata. A *monadic query* q is a function that maps an unranked tree t over Σ to a set of nodes of the tree $q(t) \subseteq \text{nodes}(t)$. If $\nu \in \text{nodes}(t)$ then we write $t(\nu)$ for the label of node ν of t in Σ .

We identify a query q with a function `query` that maps Σ -trees to Bool-trees of the same shape. It satisfies for all nodes $\nu \in \text{nodes}(t)$ that `query(t)(ν) = true` iff $\nu \in q(t)$. Let $t \in T_\Sigma$ and $\beta \in T_{\text{Bool}}$ be unranked trees of the same shape. We define the tree $t \times \beta \in T_{\Sigma \times \text{Bool}}$ such that $\text{nodes}(t \times \beta) = \text{nodes}(t)$ and $(t \times \beta)(\nu) = (t(\nu), \beta(\nu))$ for all $\nu \in \text{nodes}(t)$. Using this trick, we represent `query` as a tree language $L_{\text{query}} = \{t \times \text{query}(t) \mid t \in T_\Sigma\}$, the language of trees over $\Sigma \times \text{Bool}$ annotated by the query. We denote by π the *projection* of an annotated tree (or by extension a set of annotated trees) over Σ , i.e. $\pi(t \times \beta) = t$.

Tree languages L_{query} are *functional* in that for all trees $t \in T_\Sigma$, there exists at most one tree $\beta \in T_{\text{Bool}}$ such that $t \times \beta \in L_{\text{query}}$. Node selecting tree transducers (NSTTs) are tree automata over the signature $\Sigma \times \text{Bool}$ that recognize functional tree languages.

An example query is illustrated in Figure 5. The query `get-phone` operates on documents of the form

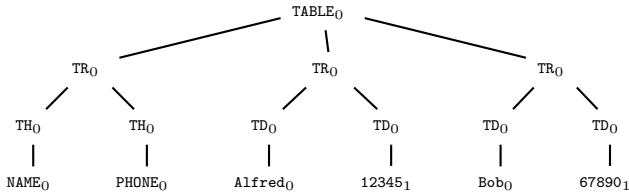


Figure 5. The tree `phone-list` and its corresponding annotated tree for query `get-phone(phone-list)`.

of `phone-list` and outputs the list of every node that contains a phone number. The application of this query over the tree `phone-list` can be represented by the tree of Figure 5 where nodes are annotated by Booleans, the value 1 is attributed to nodes extracted by the query. The query `get-phone` is confused with the language of $\Sigma \times \mathbf{Bool}$ -trees corresponding to trees annotated according to `get-phone`.

The schema-guided RPNI algorithm for monadic queries inputs a sample of examples $S \in T_{\Sigma \times \mathbf{Bool}}$ annotated by the target query and a DTD represented by a deterministic stepwise automaton D . The target query q should return $q(t) = \emptyset$ for all $t \notin L(D)$. The schema-guided RPNI algorithm for monadic queries extends over the RPNI algorithm for monadic queries defined in (Carme et al., 2007) by adding an inclusion test between $\pi(L(A))$ and $L(D)$, where A is the current hypothesis (an NSTT, i.e. a functional deterministic tree automaton over $\Sigma \times \mathbf{Bool}$). We have shown that the inclusion test can be done in $O(|A| \times |D|)$. We therefore preserve the polynomial time complexity for the schema-guided RPNI algorithm for monadic queries. The proof of correctness can be done as for the schema-guided RPNI presented in Section 2.

5. Schema-Guided Query Induction from Partial Annotations

5.1. Schema-Guided Pruning

In practice of Web information extraction, it is important to learn from trees that a user has partially annotated with respect to the expected target query. Furthermore, query learning should not be forced to recognize the whole tree even though only a small part of it is usually relevant for extraction. Pruning techniques as proposed by (Carme et al., 2007) yield solutions to both problems. Input trees are pruned in order to remove irrelevant parts. NSTTs need to be adapted to pruning, yielding pNSTTs. Those first prune subtrees non-deterministically and then select nodes from the pruned trees. This way, pNSTTs may ignore all

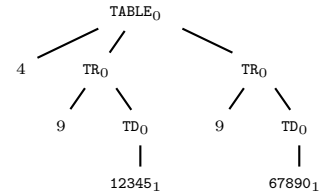


Figure 6. This tree corresponds to a possible pruning of the tree `get-phone(phone-list)` according to `schema-phone`.

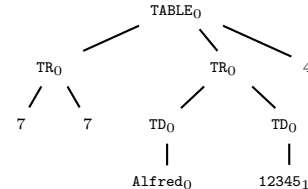


Figure 7. Another possible `schema-phone`-pruning of the tree `get-phone(phone-list)`. This tree is compatible with the tree of figure 6 regarding `schema-phone`.

pruned subtrees and operate only on relevant parts of the document.

Pruning needs to be adapted to schemas. Indeed, in pNSTTs, pruned subtrees can be replaced by any virtually tree. In a schema-guided framework we must consider that pruned subtrees are replaced by any virtually tree such that the whole tree is correct w.r.t. the schema. This is done by introducing pruning symbols which are states of the automaton representing the schema and by adapting accordingly definitions of pNSTTs.

Let schema D be a deterministic tree automaton. A pruned tree over a signature Σ with respect to schema D is a tree over the unranked signature $\Sigma_D = \Sigma \uplus \text{states}(D)$, where elements of $\text{states}(D)$ are considered as constant symbols appearing only at leaves. For every Σ -tree t , let $\text{cuts}_D(t)$ be the set of Σ_D -trees obtained by the pruning of t , i.e. obtained by substituting some complete subtrees t' of t by the corresponding state $\text{eval}_D(t')$. For instance, let us consider the schema `schema-phone` described in Section 3 by the stepwise automaton of the Figure 2. A pruned tree according to this schema will replace subtrees with the correct states. For example, trees described in Figures 6 and 7 correspond to two possible prunings of the tree `phone-list` according to `schema-phone`. Note that in (Carme et al., 2007), no schema was involved thus pruning was done using a single symbol \top . This corresponds to a pruning as defined above where the universal language is used as schema.

If you take any of the above pruned trees and replace leaves labelled with states of `schema-phone` by any subtree that evaluates in this state, you obtain a tree corresponding to a well-formed phone list. This comes from an important property of D -pruning: if $t_1 \in \text{cuts}_D(t)$ and $t \in L(D)$, then any other tree t^* such that $t_1 \in \text{cuts}_D(t^*)$ is also in $L(D)$. This holds since $t \in L(D)$ implies $\text{eval}_D(t) \in \text{final}(D)$. If you replace any subtree t' of t by another subtree t'' that evaluates in the same state, it does not change the evaluation of the other parts of the tree. This way, you can obtain the tree t^* from t , replacing subtrees of t by corresponding subtrees of t^* evaluating in the same state, without affecting the evaluation of the other part of the tree. Because of this property, we can define for any D -pruned tree t' its evaluation $\text{eval}_D(t')$ by D as being the evaluation $\text{eval}_D(t)$ of any tree t such that $t' \in \text{cuts}_D(t)$ and also denote $t' \in L(D)$ if $t \in L(D)$ and $t' \in \text{cuts}_D(t)$.

Two D -pruned trees t_1 and t_2 are said to be D -compatible if they can be obtained by D -pruning another common tree t , i.e. if $\exists t$ such that $t_1, t_2 \in \text{cuts}_D(t)$.

We now define a function to obtain pruned trees from partially annotated trees. A *partially annotated tree* over Σ is a triple $\langle t, p^+, p^- \rangle$ consisting of a Σ -tree t , a set $p^+ \subseteq \text{nodes}(t)$ of positively annotated nodes and a set $p^- \subseteq \text{nodes}(t)$ of negatively annotated nodes such that $p^+ \cap p^- = \emptyset$. It is consistent with a query `query` if $p^+ \subseteq \text{query}(t)$ and $p^- \cap \text{query}(t) = \emptyset$. The learning algorithm will be parametrized by a *pruning function*, which is a function `prune` dependent on a schema D over Σ such that $\text{prune}_D(t) \in \text{cuts}_D(t)$ for every Σ -tree.

Different pruning heuristics can be defined and the choice of the pruning function is certainly important. For example, one can use `path_extendedD` pruning function adapted from the `path_extended` strategy defined in (Carme et al., 2007). Given schema D , `path_extendedD` cuts all the nodes except those that are positively annotated, or ancestors of positively annotated nodes, or siblings of such nodes. It of course replaces all pruned subtrees by the corresponding state in D . The tree of Figure 6 has been obtained following this strategy on `get-phone(phone-list)`.

5.2. Schema-Guided Query Induction with Pruning

NSTTs are tree automata that recognize functional tree languages (for every tree t , there is at most one annotation β). As a `pNSTTD` operate on pruned trees, the notion of functionality has to be adapted. Given

a schema defined by a tree automaton D , a language L of $(\Sigma \times \text{Bool})_D$ -trees is *D -cut-functional* if for all t_1 and t_2 , which can be seen as two different D -prunings of a tree t , then the annotations on the common part of β_1 and β_2 are the same. In other words, a language of $(\Sigma \times \text{Bool})_D$ -trees is *D -cut-functional* if there is no contradiction between different prunings of the same tree.

For instance, trees of Figures 6 and 7 correspond to two `schema-phone` prunings of the tree `phone-list` which are `schema-phone-compatible`: there is no contradiction between Boolean of the two trees. If for instance the root node of the second tree would be labelled positively, the two trees would not be compatible anymore.

A pruned NSTT w.r.t. a schema represented by a tree automaton D over the signature Σ (denoted `pNSTTD`) is an NSTT over $(\Sigma \times \text{Bool})_D$ -trees whose language is *D -cut-functional*. Every `pNSTTD` A defines the query q_A such that for all Σ -trees t :

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

The schema-guided query RPNI algorithm with pruning inputs a sample of examples $S \in \mathcal{T}_{\Sigma \times \text{Bool}}$ (partially) annotated by the target query, a DTD represented by a deterministic stepwise automaton D , and a pruning function `prune` (for instance `path_extendedD` pruning function). It begins by pruning trees of the input sample according to the chosen function `prune`. Then, it extends over the RPNI algorithm for monadic queries with pruning defined in (Carme et al., 2007) in two ways: the cut-functional test is replaced by a D -cut-functionality test, and by adding an inclusion test between $\pi(L(A))$ and $L(D)$, where A is the current hypothesis `pNSTTD`. The proof of correctness can be done as for the above algorithms.

6. Conclusion

We proposed schema-guided RPNI variants for learning tree languages and monadic queries in trees. We hope that schema guidance will improve the quality of query learning algorithms in practice of Web information extraction. This is quite probable due to its immediate impact on pruning techniques. Furthermore, we hope that schema-guided techniques will render learning more independent on other heuristics. Nonetheless, it remains to validate this approach by experiments. It would be interesting to understand which properties of schemas improve the quality of query induction most in practice.

Finally, the methods presented here are limited to monadic queries. They have to be extended to the more general case of n -ary queries. In this case, when completely annotated examples are provided, the schema-guided algorithms presented here extend naturally. But the situation is more intricate in the case of partial annotations. Indeed, components of a tuple are related to each other. Therefore, it is unclear how to extend pruning techniques to this situation.

References

- Bex, G. J., Neven, F., Schwentick, T., & Tuyls, K. (2006). Inference of concise dtds from xml data. *in Proceedings of 32nd Conference on Very Large databases - VLDB* (pp. 115–126).
- Brüggemann-Klein, A., & Wood, D. (1998). One-unambiguous regular languages. *Information and Computation*, 142, 182–206.
- Carme, J., Ceresna, M., Frölich, O., Gottlob, G., Hassan, T., Herzog, M., Holzinger, W., & Krüpl, B. (2006). The lixto project: Exploring new frontiers of web data extraction. *23rd British National Conference on Databases* (pp. 1–15). Springer Verlag.
- Carme, J., Gilleron, R., Lemay, A., & Niehren, J. (2007). Interactive learning of node selecting tree transducers. *Machine Learning*, 66, 33–67.
- Carme, J., Niehren, J., & Tommasi, M. (2004). Querying unranked trees with stepwise tree automata. *19th International Conference on Rewriting Techniques and Applications* (pp. 105 – 118). Springer Verlag.
- Champavère, J., Gilleron, R., Lemay, A., & Niehren, J. (2007). Efficient inclusion checking for deterministic tree automata and xml schemas. Submitted.
- Chidlovskii, B. (2001). Wrapping web information providers by transducer induction. *Proc. European Conference on Machine Learning* (pp. 61 – 73).
- Cohen, W., Hurst, M., & Jensen, L. (2003). *Web document analysis: Challenges and opportunities*, chapter A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. World Scientific.
- Coste, F., Fredouille, D., Kermovant, C., & de la Higuera, C. (2004). Introducing domain and typing bias in automata inference. *proceedings of the 7th ICGI* (pp. 115–126). Springer Verlag.
- de la Higuera, C. (1997). Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27, 125–137.
- Denis, F., Lemay, A., & Terlutte, A. (2004). Learning regular languages using rfsas. *Theoretical Computer Science*, 313, 267–294.
- Garcia, P., & Vidal, E. (1990). Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intell.*, 12(9), 920–925.
- Gold, E. (1967). Language identification in the limit. *Inform. Control*, 10, 447–474.
- Gottlob, G., & Koch, C. (2004). Monadic datalog and the expressive power of languages for web information extraction. *Journal of the ACM*, 51, 74–113.
- Lemay, A., Niehren, J., & Gilleron, R. (2006). Learning n -ary node selecting tree transducers from completely annotated examples. *International Colloquium on Grammatical Inference* (pp. 253–267). Springer Verlag.
- Martens, W., & Niehren, J. (2007). On the minimization of XML schemas and tree automata for unranked trees. *Journal of Computer and System Science*, 73, 550–583.
- Muslea, I., Minton, S., & Knoblock, C. (2002). Active + Semi-supervised Learning = Robust Multi-view Learning. *Proceedings of ICML-2002* (pp. 435–442).
- Oncina, J., & Garcia, P. (1992). Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis* (pp. 49–61).
- Oncina, J., & García, P. (1993). *Inference of recognizable tree sets* (Technical Report). Departamento de Sistemas Informáticos y Computación, Universidad de Alicante. DSIC-II/47/93.
- Raeymaekers, S., Bruynooghe, M., & Van den Bussche, J. (2005). Learning (k,l)-contextual tree languages for information extraction. *Proceedings of ECML'2005* (pp. 305–316).