Learning Sequential Tree-to-Word Transducers

Grégoire Laurence, Aurélien Lemay, Joachim Niehren, Slawomir Staworko, Marc Tommasi

Learning Sequential Tree-to-Word Transducers

Grégoire Laurence[1,3], Aurélien Lemay[1,3], Joachim Niehren[1,4], Sławek Staworko[1,3], and Marc Tommasi[2,3]

[1] Links project, INRIA & LIFL (CNRS UMR8022)
[2] Magnet project, INRIA & LIFL (CNRS UMR8022)
[3] University of Lille
[4] INRIA, Lille

Abstract. We study the problem of learning sequential top-down tree-to-word transducers (STWs). First, we present a Myhill-Nerode characterization of the corresponding class of sequential tree-to-word transformations ($\mathcal{STW}$). Next, we investigate what learning of STWs means, identify fundamental obstacles, and propose a learning model with abstain. Finally, we present a polynomial learning algorithm.

1 Introduction

The main motivation of this paper is to study learnability of a class of tree-to-word transformations. Tree-to-word transformations are ubiquitous in computer science. They are the core of many computation paradigms from the evaluation of abstract syntactic trees to modern programming languages XSLT. For these reason, they are better suited to model general XML transformations as opposed to tree-to-tree transducers [7, 13, 14].

Following the work of [12], we study the class of deterministic sequential top-down tree-to-word transducers (STWs). STWs are finite state machines that traverse the input tree in top-down fashion and at every node produce words obtained by the concatenation of constant words and the results from processing the child nodes. STWs capture a large subclass of deterministic nested-word to word transducers (DN2W), which have recently been the object of an enlivened interest [8, 18, 19]. STWs take as an input a tree in a regular tree language and output words from a context-free grammar.

Despite of some limitations mainly due to the fact they are deterministic and top-down[5], STWs remain however very powerful. They are capable of: concatenation in the output, producing arbitrary context-free languages, deleting inner nodes, and verifying that the input tree belongs to the domain even when deleting parts of it. These features are often missing in tree-to-tree transducers, and for instance, make STWs incomparable with the class of top-down tree-to-tree transducers [7, 13]. The class of STWs has several interesting properties, in particular a normal form has been proposed in [12].

[5] non-determinism quickly leads to fundamental limitations. For instance, equivalence of non-deterministic string transducers is known to be undecidable [11]

An open question raised in [12] was the existence of a Myhill-Nerode Theorem for STWs. We solve this question and this result is the first main contribution of this paper. Myhill-Nerode Theorem provides canonical representation of languages – here, transformations – based on residuals. The Myhill-Nerode Theorem also opens a way towards grammatical inference for tree-to-word transformation. Indeed, as pointed by many authors, machine learning of formal languages is essentially a matter of estimation of equivalence classes of the target language. The second contribution is then a learning algorithm for the class of STW.

This learnability result is to be placed in a tradition of learning results for other class of grammars, starting from Gold results [10] for regular languages of words. This result has served as a basis for a host of learning algorithms including inference of regular languages of word and trees [15, 16] (see also [6] for a survey of the area), learning of DTDs and XML Schema [2, 1], XML transformations [13], and XML queries [3, 20].

The Myhill-Nerode Theorem proof starts from the identification of the class of earliest STWs (eSTWs) given in [12]. The main difficulty is to be able to characterize residual languages of a STW transformation and then define a canonical representative for eSTWs. This proof relies on an original algorithm capable to decompose a residual transformation into a form close to a rule of eSTW. In order to obtain a learning algorithm (a la RPNI [16, 15]) an important step is to decide the consistency of a transducer with a finite transformation. Unfortunately, we prove that this consistency check is NP-complete. Nevertheless, we present a learning result in a slightly modified framework where the learning algorithm can abstain from answering. We prove that we can define a characteristic sample whose cardinality is within a polynomial bound of the size of the canonical transducer of the transformation to infer. Using this last result, we present here the first polynomial time learning algorithm for the class of STW.

2 Sequential top-down tree-to-word transducers

Words and Trees For a finite set Δ of symbols, we denote by Δ^* the free monoid on Δ , i.e. the set of finite words over Δ with the concatenation operator $\cdot$ and the empty word ε. For a word u, $|u|$ is its length. For $u = u_p \cdot u_f \cdot u_s$, u_p is a *prefix* of u, u_f a *factor* of u, and u_s a *suffix* of u. The *longest common prefix* of a set of words L, denoted $lcp(L)$, is the longest word u that is a prefix of every word in L. Also, $lcs(L)$ is the *longest common suffix* of L. For $w = u \cdot v$, the *left quotient* of w by u is $u^{-1} \cdot w = v$ and the *right quotient* of w by v is $w \cdot v^{-1} = u$.

A *ranked alphabet* is a finite set of ranked symbols $\Sigma = \bigcup_{k \geq 0} \Sigma^{(k)}$, where $\Sigma^{(k)}$ is the set of k-ary symbols. We sometimes write $f^{(k)}$ to indicate explicitly that $f \in \Sigma^{(k)}$. A *tree* is a ranked ordered term over Σ. By T_Σ we denote the set of all trees over Σ. A tree language is a subset of T_Σ. A *context* C is a tree over $\Sigma \cup \{x^{(0)}\}$ with only one leaf labeled x representing a hole. By $C[t]$ we denote the tree obtained by replacing x by the tree t. A *path* is a word over $\bigcup_{k>0} \Sigma^{(k)} \times \{1, \ldots, k\}$, which identifies a node in a tree by the labels of its ancestors: ε is the root node and if a node at path p is labeled with f, then

$p \cdot (f, i)$ is the i-th child of the node. By $\mathit{paths}(t)$ we denote the set of paths of a tree t. Similarly, for a set of trees T, $\mathit{paths}(T) = \bigcup_{t \in T} \mathit{paths}(t)$. Words, trees, paths and contexts have canonical well-founded orders that are consistent with the size of object and can be tested efficiently. Using these orders, functions $\min_{Path}$, $\min_{Tree}$ and $\min_{Ctx}$ allow to obtain the minimal element of a set of resp. paths, trees or contexts.

Transducers A *deterministic sequential top-down tree-to-word transducer* (STW) is a tuple $M = (\Sigma, \Delta, Q, \mathit{init}, \delta)$, where Σ is a ranked alphabet of *input trees*, Δ is a finite alphabet of *output words*, Q is a finite set of states, $\mathit{init} \in \Delta^* \cdot Q \cdot \Delta^*$ is the *initial rule*, and δ is a partial *transition function* from $Q \times \Sigma$ to $(\Delta \cup Q)^*$ such that if $\delta(q, f^{(k)})$ is defined, then it has k occurrences of elements from Q. In the sequel, we call the state of the initial rule the *initial state*. We denote STWs the class of deterministic sequential top-down tree-to-word transducers and $\mathcal{STW}$ the class of transformations represented by an STW.

We often view δ as a set of *transition rules* , i.e. a subset of $Q \times \Sigma \times (\Delta \cup Q)^*$, which allows us to quantify over δ. Also, the transition function is extended to paths over Σ as follows: $\delta(q, \varepsilon) = q$ and $\delta(q, (f, i) \cdot p) = \delta(q_i, p)$, where $\delta(q, f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot q_k \cdot u_k$. The *size* of the STW M is the number of its states and the length of its rules, including the length of words used in the rules. The semantic of the STW M is the transformation $[\![M]\!]$ defined with the help of auxiliary transformations (for $q \in Q$) in a mutual recursion:

$$[\![M]\!]_q(f(t_1, \ldots, t_k)) = \begin{cases} u_0 \cdot [\![M]\!]_{q_1}(t_1) \cdot u_1 \cdot \ldots \cdot [\![M]\!]_{q_k}(t_k) \cdot u_k, \\ \qquad\qquad \text{if } \delta(q, f) = u_0 \cdot q_1 \cdot u_1 \ldots \cdot q_k \cdot u_k, \\ \text{undefined}, \qquad \text{if } \delta(q, f) \text{ is undefined.} \end{cases}$$

Now, $[\![M]\!](t) = u_0 \cdot [\![M]\!]_{q_0}(t) \cdot u_1$, where $\mathit{init} = u_0 \cdot q_0 \cdot u_1$. Two transducers are *equivalent* iff they define the same transformation. Also, for a transformation τ, $\mathit{dom}(\tau)$ is the domain of τ and $\mathit{ran}(\tau)$ is its range.

We also use *deterministic top-down tree automata* (DTA) that define path-closed tree languages. Recall that a tree language $L \subseteq T_\Sigma$ is *path-closed* if $L = \{t \in T_\Sigma \mid \mathit{paths}(t) \subseteq \mathit{paths}(L)\}$. We refer the reader to [5] for a precise definition and point out that a DTA is essentially an STW with empty output alphabet thus defining a constant transformation mapping every element of its domain to the empty word.

Earliest Transducers The construction of the canonical transducer, the core of the Myhill-Nerode characterisation of $\mathcal{STW}$, is inspired by the normal form for STWs. The usual choice to define normal forms of transducers is to produce the output as early as possible. This idea initially comes from normalisation of word-to-word transducers, as in [4], and is also employed in [13, 9] for tree-to-tree transducers. In [12], we have proposed the following normal form for STWs. An STW $M = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ is *earliest* (eSTW) iff:

($\mathbf{E_1}$) $\mathit{lcp}(\mathit{ran}([\![M]\!]_q)) = \varepsilon$ and $\mathit{lcs}(\mathit{ran}([\![M]\!]_q)) = \varepsilon$ for every state q,

($\mathbf{E_2}$) for $init = u_0 \cdot q_0 \cdot u_1$, $lcp(ran([\![M]\!]_{q_0}) \cdot u_1) = \varepsilon$ and for $\delta(q, f) = u_0 \cdot q_1 \cdot \ldots \cdot q_k \cdot u_k$, then $\forall 1 \leq i \leq k$, $lcp(ran([\![M]\!]_{q_i}) \cdot u_i \cdot \ldots \cdot ran([\![M]\!]_{q_k}) \cdot u_k) = \varepsilon$. □

Essentially, **($\mathbf{E_1}$)** ensures that the output is produced as *up* as possible during the parsing while **($\mathbf{E_2}$)** ensures output is produced as *left* as possible. We also observe that in an eSTW, transformations $[\![M]\!]_q$ associated with states have the property that the *lcp* an *lcs* of their output is empty. It is know that for every STW there exists a unique minimal equivalent eSTW [12].

Example 1. Consider the transformation τ_1 that takes as an input a tree t over the signature $\Sigma = \{f^{(2)}, a^{(0)}, b^{(0)}\}$ and output a word on $\Delta = \{\#\}$ that counts the number of symbols in t (i.e. $\tau_1(f(f(a, b), a)) = \#\#\#\#\#$). This can be done by transducer $M_1 = (\Sigma, \Delta, Q_1 = \{q\}, init_1 = q, \delta_1)$ with $\delta_1(q, a) = \delta_1(q, b) = \#$ and $\delta_1(q, f) = q \cdot \# \cdot q$. However M_1 is not earliest: the output always starts with an # which can be produced earlier (**($\mathbf{E_1}$)** is not satisfied), and the symbol # in the rule $\delta_1(q, f)$ could be produced before the states (**($\mathbf{E_2}$)** is not satisfied).

Consider $M_1' = (\Sigma, \Delta, Q_1' = \{q\}, init_1' = \# \cdot q, \delta_1')$ with $\delta_1'(q, a) = \delta_1'(q, b) = \varepsilon$ and $\delta_1'(q, f) = \#\# \cdot q \cdot q$. This transducer also represent τ_1 but is earliest.

3 A Myhill-Nerode Theorem for $\mathcal{STW}$

In this section we present the construction of a canonical eSTW $Can(\tau)$ that captures an arbitrary $\mathcal{STW}$ transformation τ. Because STWs process the input tree in a top-down fashion, we shall decompose τ into several transformations that capture the transformation performed by τ on the children of the input tree. The decomposition is then used to recursively define the notion of residual $p^{-1}\tau$ of τ w.r.t. a path p, essentially the transformation performed by τ at the node reached with p of its input tree. Residuals are used to define in the standard way the Myhill-Nerode equivalence relation and the canonical transducer $Can(\tau)$.

Decomposition We fix a transformation τ and let $Left(\tau) = lcp(ran(\tau))$ and $Right(\tau) = lcs(Left(\tau)^{-1} ran(\tau))$. τ is *reduced* if $Left(\tau) = Right(\tau) = \varepsilon$. The *core* of τ is defined as $Core(\tau) = \{(t, Left(\tau)^{-1} \cdot w \cdot Right^{-1}(\tau)) \mid (t, w) \in \tau\}$. While not every transformation is reduced, its core is and it preserves the essence of the original transformation .

A *decomposition* of a reduced τ for $f \in \Sigma$ is a sequence $u_0\tau_1u_1 \ldots u_{k-1}\tau_k u_k$, where $u_0, \ldots, u_k$ are words and $\tau_1, \ldots, \tau_k$ transformations, that satisfy the following natural conditions:

($\mathbf{D_1}$) $dom(\tau_i) = \{t_i \mid f(t_1, \ldots, t_k) \in dom(\tau)\}$,
($\mathbf{D_2}$) $\forall t = f(t_1, \ldots, t_k) \in dom(\tau)$, $\tau(t) = u_0\tau_1(t_1)...\tau_k(t_k)u_k$,

and to ensure the uniqueness of decomposition we impose two additional conditions that are obtained by reformulation of **($\mathbf{E_1}$)** and **($\mathbf{E_2}$)**:

($\mathbf{C_1}$) τ_i is reduced, and
($\mathbf{C_2}$) $lcp(ran(\tau_i) \cdot u_i \cdot \ldots \cdot ran(\tau_k) \cdot u_k) = \varepsilon$.

We point out that not every transformation can be decomposed.

Example 2. Take $\tau_{\text{swap}} = \{(f(a,a), aa),\ (f(a,b), ba),\ (f(b,a), ab),\ (f(b,b), bb)\}$ that outputs leaves in reverse order. This transformation can not be performed by an STW and there is no decomposition for it.

Residuals The *residual* of a transformation τ at a path p is defined recursively: $\varepsilon^{-1}\tau = \mathit{Core}(\tau)$ and $(p \cdot (f,i))^{-1}\tau = \tau_i$ if $u_0 \cdot \tau_1 \dots \tau_n \cdot u_n$ is the unique decomposition of $p^{-1}\tau$ for f.

Example 3. Consider the transformation τ_1 of example 1. $\mathit{Core}(\tau_1) = \tau_1'$ where τ_1' gives the number of symbol in t minus one. Then $\varepsilon^{-1}\tau_1 = \tau_1'$. Also, The decomposition of τ_1' for f is $\#\# \cdot ((f,1)^{-1}\tau_1) \cdot ((f,2)^{-1}\tau_1)$. Observe that $(f,1)^{-1}\tau_1 = (f,2)^{-1}\tau_1 = \tau_1'$, so this decomposition is in fact $\#\# \cdot \tau_1' \cdot \tau_1'$. Also, $\mathit{lcp}(\tau_1') = \mathit{lcs}(\tau_1') = \varepsilon$. This decomposition is consistent with the rules of M_2.

Example 4. Consider the transformation τ_2 that takes as an input a tree $t = f(t_1, t_2)$ over $\Sigma = \{f^{(2)}, a^{(0)}, b^{(0)}\}$ and output a word over $\Delta = \{\#\}$ such that the number of $\#$ is equal to the number of f and a symbols of t_1 plus the number of f and b symbols of t_2, e.g. $\tau_2(f(f(a,b),b))) = \#^3$ (2 $\#$ for $f(a,b)$ and one for b). As τ_2 is reduced, $\varepsilon^{-1}\tau_2 = \mathit{Core}(\tau_2) = \tau_2$. The decomposition of τ for f is $\tau_3 \cdot \tau_4$ with $\tau_3(a) = \#$, $\tau_3(b) = \varepsilon$. The decomposition of τ_3 at f is $\# \cdot \tau_3 \cdot \tau_3$. Similarly, $\tau_4(a) = \varepsilon$, $\tau_4(b) = \#$, and its decomposition at f is $\# \cdot \tau_4 \cdot \tau_4$.

Naturally, not every transformation has well-defined residuals. However any $\mathcal{STW}$ transformation has them and there is a strict correspondence between the residuals and the states of an eSTW defining the transformation.

Lemma 5. *Let M be an* eSTW *with initial state q_0. For any $p \in \mathit{paths}(\mathit{dom}([\![M]\!]))$ we have $p^{-1}[\![M]\!] = [\![M]\!]_{\delta(q_0,p)}$.*

This result suggest, and we proved it later on, that the existence of residuals of a transformation for any path of its domain is an important necessary condition for being $\mathcal{STW}$. Consequently, we say that τ is *sequential top-down* if and only if $p^{-1}\tau$ exists for every $p \in \mathit{paths}(\mathit{dom}(\tau))$.

Canonical transducer Having defined residuals the construction of the canonical transducer $\mathit{Can}(\tau)$ for a transformation τ is standard. The *Myhill-Nerode equivalence relation* $\equiv_\tau$ on paths of τ is defined in the standard manner: $p_1 \equiv_\tau p_2$ iff $p_1^{-1}\tau = p_2^{-1}\tau$ for $p_1, p_2 \in \mathit{paths}(\mathit{dom}(\tau))$. The Myhill-Nerode equivalence class of a path p w.r.t. τ is $[p]_\tau = \{p' \in \mathit{paths}(\mathit{dom}(\tau)) \mid p \equiv_\tau p'\}$. We say that τ has finite Myhill-Nerode index if $\equiv_\tau$ has a finite number of equivalence classes.

The *canonical transducer* $\mathit{Can}(\tau) = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ of a sequential top-down transformation τ of finite Myhill-Nerode index follows: 1) the set of states is $Q = \{[p]_\tau \mid p \in \mathit{paths}(\mathit{dom}(\tau))\}$; 2) the initial rule is $\mathit{init} = \mathit{Left}(\tau) \cdot [\varepsilon]_\tau \cdot \mathit{Right}(\tau)$; 3) for every state $[p] \in Q$ and every $f \in \Sigma$ such that $p^{-1}\tau$ has a decomposition $u_0 \cdot \tau_1 \cdot u_1 \dots \tau_k \cdot u_k$ for f, the canonical transducer $\mathit{Can}(\tau)$ has the transition rule $\delta([p], f) = u_0 \cdot [p \cdot (f,1)]_\tau \cdot u_1 \cdot \dots \cdot u_{k-1} \cdot [p \cdot (f,k)]_\tau \cdot u_k$.

Theorem 6. *For any transformation τ the following conditions are equivalent: 1) τ is definable by an* STW*; 2) τ is sequential top-down and has a finite Myhill-Nerode index; 3) $Can(\tau)$ is the unique minimal* eSTW *defining τ.*

Direction (1) to (2) requires normalizing the STW into an eSTW and using Lemma 5. Point (3) is obtained from (2) by establishing that the minimal eSTW is in fact $Can(\tau)$ (modulo state renaming). Direction (3) to (1) is trivial.

Example 7. The canonical transducer of τ_2 (as in example 4) is $Can(\tau_2) = M_2$ defined as follow: $M_2 = (\Sigma, \Delta, Q = \{q_1, q_2, q_3\}, init = q_1, \delta)$ with $\delta(q_1, f) = \# \cdot q_2 \cdot q_3$, $\delta(q_2, f) = \# \cdot q_2 \cdot q_2$, $\delta(q_2, a) = \#$, $\delta(q_2, b) = \varepsilon$, $\delta(q_3, f) = \# \cdot q_3 \cdot q_3$, $\delta(q_3, a) = \varepsilon$ and $\delta(q_3, b) = \#$. This is consistent with decompositions observed in example 4 if one identifies q_1 with ε, q_2 with $(f, 1)$ and q_3 with $(f, 2)$.

4 Learning STWs

In this section we present a learning algorithm for $\mathcal{STW}$ transformations.

4.1 Learning framework

First, we investigate the question of the meaning of what learning a transformation means and pursue an answer that is inspired by the Gold learning model in polynomial time and data [10]. Essentially, we are interested in a polynomial time algorithm that takes a finite *sample* $S \subseteq T_\Sigma \times \Delta^*$ and constructs an STW M transducer *consistent* with S i.e., $S \subseteq [\![M]\!]$. Unfortunately, unless P = NP, the following result precludes the existence of such an algorithm.

Theorem 8. *Checking if there exists an* STW *consistent with a given sample is NP-complete.*

To overcome this difficulty, we shall allow the algorithm to *abstain* i.e., return a special *Null* for cases when an STW consistent with the input sample cannot be easily constructed. Naturally, this opens the door to a host of trivial algorithms that return *Null* for all but a finite number of hard-coded inputs. To remove such trivial algorithms from consideration we shall essentially require that the learning algorithm of interest can infer any $\mathcal{STW}$ τ from sufficiently informative samples, called *characteristic sample* of τ: the learning algorithm should be able to output an eSTW defining τ. Furthermore, we require the characteristic sample to use a number of examples bounded by a polynomial of the number of equivalence classes of $\equiv_\tau$.

Another obstacle comes from the fact that DTAs are not learnable from positive examples alone and learning DTA from a set of positive examples can be easily reduced to learning STW. To remove this obstacle, we assume that a DTA D capturing the domain of the goal transformation is given on the input. Note that this domain automaton could also be obtained by learning method, such as the RPNI algorithm for trees [15].

If a class of transformation satisfies all the above properties, we say that it is learnable with abstain from polynomial time and data. In the following, we aim to obtain the following result.

Theorem 9. *$\mathcal{STW}$ transformations represented by* eSTW *are learnable with abstain from polynomial time and data.*

4.2 Learning Algorithm

We now present the learning algorithm for $\mathcal{STW}$. This algorithm essentially attempts to emulate the construction of the canonical transducer, using a finite sample of the transformation.

The Core Algorithm The main procedure of the learning algorithm follows closely the construction of the canonical transducer. It takes as an input a sample S of a target transformation τ, as well as a DTA D that represents $dom(\tau)$.

Algorithm 1 $\mathsf{learner}_D(S)$

1: $P := paths(dom(S))$; $Q := \emptyset$
2: $state :=$ **new** $hashtable\langle Path, Path\rangle()$
3: **while** $P \neq \emptyset$ **do**
4: $\quad p := \min_{Path}(P)$
5: $\quad P' := \{p' \in Q \mid p \simeq_{S,D} p'\}$
6: $\quad$ **if** $P' \neq \emptyset$ **then** (*p *can be merged**)
7: $\quad\quad P := P \setminus \{p' \in P \mid p$ is prefix of $p'\}$
8: $\quad\quad state[p] := \min_{Path}(P')$
9: $\quad$ **else**
10: $\quad\quad P := P \setminus \{p\}$; $Q := Q \cup \{p\}$
11: $\quad\quad state[p] := p$
12: $init := Left(S) \cdot state[\varepsilon] \cdot Right(S)$
13: **for** $p \in Q$ **do**
14: $\quad$ **for** $f \in \Sigma$ s.t. $\exists i$ with $p.(f,i) \in paths(dom(S))$ **do**
15: $\quad\quad$ **for** $i \in 1, ...k$, Let $p_i = state[p.(f,i)]$
16: $\quad\quad (u_0, \_, u_1, \ldots, u_k) := \mathsf{decomp}(\mathsf{residual}(S,p), f)$
17: $\quad\quad \delta(p,f) := u_0 \cdot p_1 \cdot u_1 \cdot \ldots \cdot p_k \cdot u_k$
18: $M := (\Sigma, \Delta, Q, init, \delta)$
19: **if** $S \subseteq [\![M]\!]$ **and** $dom([\![M]\!]) \subseteq [\![D]\!]$ **then return** M **else return** *Null*

The algorithm consists of 2 parts. First, in lines 3 to 11, it attempts to identify the set of states of the canonical transducer. For this, it builds a function *state* that associates with every path the minimal path in its equivalence class that represents the corresponding residual. This is based on the predicate $\simeq_{S,D}$ which is an emulation of the Myhill-Nerode equivalence relation $\equiv_\tau$ on an finite sample of τ. Note that if $\simeq_{S,D}$ behaves exactly as $\equiv_\tau$, and assuming $paths(dom(S))$ contains all smallest paths representative of each residual, this procedure produces exactly

the set Q of states of $Can(\tau)$. The exact implementation of the predicate $\simeq_{S,D}$ is explained later.

Second part, line 12, builds the other elements of the transducer. This uses the procedure decomp to compute decomposition of samples in a manner emulating decomposition of transformations and is explained in detail later.

We point out the algorithm may fail to produce an eSTW consistent with S. Therefore, in line 19 the consistence of the constructed eSTW is verified and the algorithm abstains from answer if the test fails. The following lemma is therefore trivial.

Lemma 10. *For a sample S and a* DTA *D, $\mathsf{learner}_D(S)$ produces an* eSTW *M in time polynomial in the size of S or abstains from answer.*

This results assumes the existence of polynomial procedures for $\simeq_{S,D}$, decomp and $\mathsf{residual}$, which we present next.

Decomposition The above learning algorithm relies on the ability to decompose a sample. This is done by the following procedure. It takes as an input a sample S which is supposed to be representative of a transformation τ, and a symbol $f^{(}k)$ such that there are f rooted trees in S. From this, it outputs a sequence $u_0 \cdot S_1 \cdot u_1 \dots S_k \cdot u_k$ which ideally is the proper decomposition of S w.r.t. to τ.

Algorithm 2 $\mathsf{decomp}(S, f^{(k)})$

1: Let $S_f = \{(t, w) \in S \mid t \text{ is of the form } f(t_1, \dots t_k)\}$
2: Let $s = f(s_1, \dots, s_k)$ be the tree $\min_{Tree}(dom(S_f))$ and $w_s := S(s)$
3: **for** $i := 1, \dots, k$ **do** $D_i := \{t_i \mid f(s_1, \dots, s_{i-1}, t_i, s_{i+1}, \dots, s_k) \in dom(S_f)\}$
4: $u_0 := lcp(\{w \mid (t, w) \in S_f\})$
5: $\mathit{prefix}_0 = u_0$
6: **for** $i := 1, \dots, k$ **do**
7: $\quad \mathit{prefix}_i := lcp\{w \mid \exists t_{i+1}, \dots, t_k.\ (f(s_1, \dots, s_i, t_{i+1}, \dots, t_k), w) \in S_f\}$
8: $\quad \mathit{suffix}_i := \mathit{prefix}_i^{-1} \cdot w_s$
9: $\quad S'_i := \emptyset$
10: $\quad$ **for** $t \in D_i$ **do**
11: $\quad\quad w := \mathit{prefix}_{i-1}^{-1} \cdot S(f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_k)) \cdot \mathit{suffix}_i^{-1}$
12: $\quad\quad S'_i := S'_i \cup \{(t, w)\}$
13: $\quad u_i := lcs(ran(S'_i))$
14: $\quad S_i := \{(t, w \cdot u_i^{-1}) \mid (t, w) \in S'_i\}$
15: **return** $(u_0, S_1, u_1 \dots, S_k, u_k)$

From the minimal tree $s = f(s_1, \dots, s_k)$ of $dom(S)$ rooted by f, the algorithm essentially tries to decompose $w_s = S(s)$ into $u_0 S_1(s_1) \dots S_k(s_k) u_k$, as defined by the formal definition of $\mathsf{decomp}(S, f)$. Note this is defined only if there are some f rooted trees in $dom(S)$. The word u_0 is simply $\mathit{Left}(S_f)$. Then, for each i, prefix_i is built such that it is equal to $u_0 S_1(s_1) \dots S_i(s_i) u_i$ and so $\mathit{suffix}_i = \mathit{prefix}_i^{-1} w_s = S_{i+1}(s_{i+1}) \dots u_k$. From this, residual transformations S_i and words u_i can be built simultaneously. For any tree $t_i \in (f, i)^{-1} dom(S)$, we consider the tree

$t = f(s_1, \ldots, s_{i-1}, t_i, s_{i+1}, \ldots, s_k))$ (which belongs to $dom(S)$ if is path-closed or well constructed) and compute $S'_i(t_i) = S_i(t_i) \cdot u_i = \mathit{prefix}_{i-1}^{-1} S(t) \mathit{suffix}_i^{-1}$. The word u_i is obtained as $lcs(ran(S'_i))$, which allow to obtain $S_i(t_i) = S'_i(t_i) \cdot u_i^{-1}$.

If the sample is rich enough (a notion that will be made precise in the next section), the *lcp* and *lcs* of the different elements are computed correctly and the algorithm outputs exactly what it supposed to. If the sample is not rich enough, it may possibly produce a decomposition which is not necessarily sound: there may be a tree $t = f(t_1, \ldots, t_k)$ such that which $S(t) \neq u_0 \cdot S_1(t_1) \cdot u_1 \ldots S_k(t_k) \cdot u_k$. However, in any case, the algorithm answers in time polynomial in the size of S.

Residuals and Equivalence From the decomposition procedure, it is possible to build the residual of a sample for a path p. $\mathsf{residual}(S, p)$ is computed in a manner analogous to $p^{-1}\tau$: for $p = \varepsilon$, $\mathsf{residual}(S, p) = \mathsf{reduce}(S)$, and for $p = p' \cdot (f, i)$, we compute $S' = \mathsf{residual}(S, p)$ and $\mathsf{residual}(S, p) = S_i$, where $\mathsf{decomp}(S', f) = u_1 \cdots S_1 \ldots S_k \cdot u_k$. Note that again, $\mathsf{residual}(S, p)$ is a polynomial time procedure.

From this, we can define the relation $\simeq_{S,D}$ which tries to emulate $\equiv_\tau$. Recall that $p_1 \equiv_\tau p_2$ iff $p_1^{-1}\tau = p_2^{-1}\tau$ and note that two transformations are identical if they have the same domain and agree on every tree. Because the residuals $p_1^{-1}\tau$ and $p_2^{-1}\tau$ are represented with finite samples $S_1 = \mathsf{residual}(S, p_1)$ and $S_2 = \mathsf{residual}(S, p_2)$ and their domains need not be necessarily equal, the predicate $p_1 \simeq_{S,D} p_2$ uses the DTA D to verify that the domains of the residuals $p_1^{-1}\tau$ and $p_2^{-1}\tau$ are equal and then checks that for every tree in common both samples S_1 and S_2 produce the same results.

Again, all those procedures are polynomial. Note however that they behave correctly (i.e. $p \simeq_{S,D} p' \Leftrightarrow p \equiv_\tau p'$ for instance) only if the sample is rich enough. What it means exactly is defined in the next section.

4.3 A Characteristic Sample

In the following, we identify a characteristic sample for STW transformation τ: $\mathit{CharSet}(\tau)$ is a finite set of examples such that whenever $\mathsf{learner}$ is provided a superset of $\mathit{CharSet}(\tau)$ as input, it outputs $\mathrm{can}(\tau)$.

The Characteristic Sample We first introduce some notations and definitions. For $p \in \mathit{paths}(\mathit{dom}(\tau))$ let c_p be the minimal context with x at path p. The finite set of all minimal representatives of equivalence classes of $\equiv_\tau$ is $\mathit{StatePath}(\tau) = \{\min_{Path}([p]_\tau) \mid p \in \mathit{paths}(\mathit{dom}(\tau))\}$. We also define $\mathit{EdgePath}(\tau)$, which adds to the shortest paths their extensions with one additional step i.e., $\mathit{EdgePath}(\tau) = \mathit{StatePath}(\tau) \cup \{p \cdot (f, i) \in \mathit{paths}(\mathit{dom}(\tau)) \mid p \in \mathit{StatePath}(\tau)\}$.

Example 11. τ_2 has 3 distincts residuals: $\varepsilon^{-1}\tau_2$, $(f, 1)^{-1}\tau_2$ and $(f, 2)^{-1}\tau$. Therefore, $\mathit{StatePath}(\tau_2) = \{\varepsilon, (f, 1), (f, 2)\}$ and $\mathit{EdgePath}(\tau_2) = \mathit{StatePath}(\tau_2) \cup \{(f, 1)(f, 1), (f, 1)(f, 2), (f, 2)(f, 1), (f, 2)(f, 2)\}$.

Let us consider a path $p \in \mathit{EdgePath}(\tau)$, and a set of trees $T \subseteq T_\Sigma$. Then, T is *structurally representative* for τ with respect to p if

(S$_0$) the tree $\min_{Tree}(dom(p^{-1}\tau))$ belongs to T;
(S$_1$) $lcp((p^{-1}\tau)(T)) = \varepsilon$ and $lcs((p^{-1}\tau)(T)) = \varepsilon$;
(S$_2$) $lcp(ran(p^{-1}\tau) \setminus \{\varepsilon\}) = lcp((p^{-1}\tau)(T) \setminus \{\varepsilon\})$.

Additionally, we say that T is *discriminant* for τ with respect to p if

(DI) for any $p_0 \in StatePath(\tau)$, if $T_{p,p_0} = \{t \in dom(p^{-1}\tau) \cap dom(p_0^{-1}\tau) \mid p^{-1}\tau(t) \neq p_0^{-1}\tau(t)\}$ is nonempty, then $\min_{Tree}(T_{p,p_0})$ belongs to T.

For a path p, conditions **(S$_0$)**, **(S$_1$)** and **(S$_2$)** ensure that T contains all elements needed to correctly decompose the residual transformation $p^{-1}\tau$. Condition **(DI)** ensures that T contains witnesses necessary to distinguish different equivalence classes.

Example 12. Consider transformation τ_2 and take for instance $p = (f, 1)$. The tree $T_{p,\varepsilon}$ is the smallest tree whose image differs in $p^{-1}\tau_2$ and $\varepsilon^{-1}\tau_2$. In fact, $T_{p,\varepsilon} = f(a, a)$ as $p^{-1}\tau_2(f(a, a)) = \#^2$ and $\varepsilon^{-1}\tau_2(f(a, a)) = \#^3$. For other $p' \in \{(f, 2), (f, 2)(f, 1), (f, 2)(f, 2)\}$, $T_{p,p'} = a$.

To satisfy condition **(S$_1$)** and **(S$_2$)**, one can take $\{a, b, f(a, a)\} \in T_p$. This allows to satisfy **(S$_1$)** as $lcp(\{(p^{-1}\tau_2)(a), (p^{-1}\tau_2)(b), (p^{-1}\tau_2)(f(a, a))\}) = lcp(\{\#, \varepsilon, \#^3\}) = \varepsilon$ and the same for *lcs*. For **(S$_2$)**, we have $lcp(\{(p^{-1}\tau_2)(a), (p^{-1}\tau_2)(b), (p^{-1}\tau_2)(f(a, a))\} \setminus \{\varepsilon\}) = lcp(\{\#, \varepsilon, \#^3\} \setminus \{\varepsilon\}) = \#$ which is indeed equal to $lcp(ran(p^{-1}\tau) \setminus \{\varepsilon\})$.

Let τ be a transformation in $\mathcal{STW}$ and let p be a path in $EdgePath(\tau)$. A sample S is characteristic for τ at path p if (i) $S \subseteq p^{-1}\tau$ and ; (ii) for all paths p_0 such that $p \cdot p_0 \in EdgePath(\tau)$, the set of trees $c_{p_0}^{-1} dom(S)$ is discriminant and structurally representative for τ with respect to $p \cdot p_0$. A sample is *characteristic* for τ if it is characteristic for τ at path ε.

An important property is that it is possible to build a characteristic sample whose cardinality is with a polynomial bound on the number of distinct residuals of τ. Indeed, to have property **(DI)**, one need a quadratic number of trees while conditions **(S$_0$)**, **(S$_1$)**, and **(S$_2$)** all require a linear number of trees. We denote by $CharSet(\tau, p)$ the minimal characteristic sample for τ at path p and by $CharSet(\tau)$ the set $CharSet(\tau, \varepsilon)$. This yields the following lemma.

Lemma 13. *For any* eSTW M *there exists a characteristic sample* $CharSet(\llbracket M \rrbracket)$ *of cardinality polynomial in the size of* M.

We also point out that any sample S consistent with $\llbracket M \rrbracket$ that contains $CharSet(\llbracket M \rrbracket)$ is also characteristic for $\llbracket M \rrbracket$.

Example 14. From previous example, one can build a characteristic sample for τ. In particular, the minimal context for $(f, 1)$ is $f(x, a)$. In example 12, it is argued that trees $\{a, b, f(a, a)\}$ are in T_p, which means that $CharSet(\tau)$ contains $(f(a, a), \#^3)$, $(f(b, a), \#^2)$ $(f(f(a, a), a), \#^5)$. A similar approach has to be also considered for all other elements of $EdgePath(\tau)$ to obtain the full $CharSet(\tau)$.

Decomposition of Characteristic Samples It remains to see that from the characteristic sample of a transduction, the procedures used by the learning algorithm behave as expected. We begin with the decomposition. The first lemma shows that the factors of a decomposition are identified whenever a superset of the characteristic sample is provided to the decomposition procedure.

Lemma 15. *Let $\tau \in \mathcal{STW}$ and $p \in \mathit{StatePath}(\tau)$. Let S be a characteristic set for τ at path p, For any $f \in \Sigma^{(k)}$ such that the decomposition of $p^{-1}\tau$ at f is $u_0 \cdot \tau_1 \dots \tau_k \cdot u_k$, then $\mathsf{decomp}(S, f) = u_0 \cdot S_1 \dots S_k \cdot u_k$ where each S_i is characteristic for τ at path $p \cdot (f, i)$*

This decomposition lemma relies on the idea that the properties required by the formal definition can be observed locally on a characteristic sample: for instance property $\mathbf{(D_1)}$ and $\mathbf{(D_2)}$ simply comes from consistency of the sample ($S \subseteq \tau$), while $\mathbf{(C_1)}$ is observable on S thanks to property $\mathbf{(S_1)}$. However, $\mathbf{(C_2)}$ does not translate directly into a property that a characteristic sample should fulfill. This is of course the role played by property $\mathbf{(S_2)}$.

The link between $\mathbf{(S_2)}$ and $\mathbf{(C_2)}$ is actually an indirect consequence of following property: let W and W' be two sets of words in Δ^*, if $\mathit{lcp}(W \setminus \{\varepsilon\}) = \mathit{lcp}(W' \setminus \{\varepsilon\})$, and $\mathit{lcp}(W) = \mathit{lcp}(W')$, then $\mathit{lcp}(\{w \cdot u \mid w \in W\}) = \varepsilon$ for a $u \in \Delta^*$ implies that $\mathit{lcp}(\{w' \cdot u \mid w' \in W'\}) = \varepsilon$.

Now, consider a transformation $\tau \in \mathcal{STW}$, a path $p \in \mathit{StatePath}(\tau)$ and a sample S characteristic for τ in a path p. If we consider $\mathsf{decomp}(p^{-1}\tau, f) = u_0\tau_1 \dots \tau_k u_k$, then for any $i \in \{1, \dots, k\}$ we have $\mathit{lcp}\{\tau_i(t_i) \cdot u_i \cdot \dots \cdot \tau_k(t_k) \cdot u_k \mid t_i \in (f, i)^{-1}\mathit{dom}(S), \dots, t_k \in (f, k)^{-1}\mathit{dom}(S)\} = \varepsilon$.. This is a direct consequence of above property and the fact that S satisfy $\mathbf{(S_1)}$ and $\mathbf{(S_2)}$, and allows us to prove Lemma 15.

As the construction of residuals $\mathsf{residual}(S, p)$ relies on the decomposition, Lemma 15 has the important consequence that those residuals can be computed properly for any $p \in \mathit{EdgePath}(\tau)$. This gives the following two results. First, if S is characteristic for τ, and $p \in \mathit{EdgePath}(\tau)$, then $\mathsf{residual}(S, p)$ is characteristic for τ w.r.t. p. Second, as a consequence and because of $\mathbf{(DI)}$, if $p, p' \in \mathit{EdgePath}(\tau)$ then $p \simeq_{S,D} p' \Leftrightarrow p \equiv_\tau p'$. Ultimately, this indicates that from a sample S characteristic for τ, the learning algorithm builds $\mathit{Can}(\tau)$:

Lemma 16. *Let $\tau \in \mathcal{STW}$ and D a* DTA *with $[\![D]\!] = \mathit{dom}(\tau)$. From any sample S characteristic with τ, $\mathsf{learner}_D(S) = \mathit{Can}(\tau)$.*

This, along with Lemmas 10 and 13 proves Theorem 9.

5 Conclusion

We presented the first polynomial time learning algorithm for tree to string transformation. This algorithm present the particularity to abstain answering at some point. This is due to the fact that the consistency problem is NP-complete for STW, and so, it is simply not possible to provide a transducer consistant with some input sample.

Also note that the language of strings outputed by an STW are context free languages. Therefore, inference of STW is linked to inference of Context Free Grammars (CFG) and can be seen as the inference of a CFG using words and their derivative trees as input. This work may therefore bring some highlight to the problem of Context Free Grammar inference.

References

1. G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *ACM Transactions on the Web*, 4(4), 2010.
2. G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM TODS 2010*, 35(2).
3. J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1):33–67, 2007.
4. C. Choffrut. Minimizing subsequential transducers: a survey. *TCS*, 292(1):131–143, 2003.
5. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available online since 1997: http://tata.gforge.inria.fr, October 2007.
6. C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38:1332–1348, September 2005.
7. J. Engelfriet, S. Maneth, and H. Seidl. Deciding equivalence of top-down XML transformations in polynomial time. *Journal of Computer and System Science*, 75(5):271–286, 2009.
8. E. Filiot, J.F. Raskin, P.A. Reynier, F. Servais, and J.M. Talbot. Properties of visibly pushdown transducers. In *MFCS 2010*, p. 355–367.
9. S. Friese, H. Seidl, and S. Maneth. Minimization of deterministic Bottom-Up tree transducers. In *DLT 2010*, vol. 6224 of *LNCS*, 185–196. 2010.
10. E. M. Gold. Complexity of automaton identification from given data. *Inform. Control*, 37:302–320, 1978.
11. T. V. Griffiths. The unsolvability of the equivalence problem for Lambda-Free nondeterministic generalized machines. *Journal of the ACM*, 15(3):409–413, 1968.
12. G. Laurence, A. Lemay, J. Niehren, S. Staworko, and M. Tommasi. Normalization of sequential Top-Down Tree-to-Word transducers. In *LATA 2011*, p. 354–365.
13. A. Lemay, S. Maneth, and J. Niehren. A learning algorithm for Top-Down XML transformations. In *29th PODS 2010*, pages 285–296. ACM-Press, 2010.
14. W. Martens, F. Neven, and M. Gyssens. Typechecking top-down XML transformations: Fixed input or output schemas. *Inf. Comput.*, 206(7):806–827, 2008.
15. J. Oncina and P. García. Inference of recognizable tree sets. Tech. report, Dept de Sistemas Informáticos y Computación, Univ. de Alicante, 1993. DSIC-II/47/93.
16. J. Oncina and P. Gracia. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, pages 99–108, 1992.
17. C. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
18. J-F. Raskin and F. Servais. Visibly pushdown transducers. In *Automata, Languages and Programming 2008*, p. 386–397.
19. S. Staworko, G. Laurence, A. Lemay, and J. Niehren. Equivalence of nested word to word transducers. In *FCT 2009*, p. 310–322.
20. S. Staworko and P. Wieczorek. Learning XML twig queries. *CoRR*, abs/1106.3725, 2011.