Normalization of Sequential Top-Down Tree-to-Word Transducers

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

[1] Mostrare project, INRIA & LIFL (CNRS UMR8022)
[2] University of Lille, France
[3] INRIA, Lille, France

Abstract. We study normalization of deterministic sequential top-down tree-to-word transducers (STWs), that capture the class of deterministic top-down nested-word to word transducers. We identify the subclass of *earliest* STWs (eSTWs) that yield unique normal forms when minimized. The main result of this paper is an effective normalization procedure for STWs. It consists of two stages: we first convert a given STW to an equivalent eSTW, and then, we minimize the eSTW.

1 Introduction

The classical problems on transducers are equivalence, minimization, learning, type checking, and functionality [2,13,14,6]. Except for the latter two questions, one usually studies deterministic transducers because non-determinism quickly leads to fundamental limitations. For instance, equivalence of non-deterministic string transducers is known to be undecidable [8]. We thus follow the tradition to study classes of deterministic transducers. The problems of equivalence, minimization, and learning are often solved using unique normal representation of transformations definable with a transducer from a given class [9,7,5,11]. Normalization i.e., constructing the normal form of a given transducer, has been studied independently for various classes, including string transducers [4,3], top-down tree transducers [5], and bottom-up tree transducers [7].

In this paper, we study the normalization problem for 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. The main motivation to study this model is because tree-to-word transformations are better suited to model general XML transformations as opposed to tree-to-tree transducers [5,11,14]. This follows from the observation that general purpose XML transformation languages, like XSLT, allow to define transformations from XML documents to arbitrary, not necessarily structured, formats. Also, STWs capture a large subclass of deterministic nested-word to word transducers (DN2W), which have recently been the object of an enlivened interest [6,15,16].

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 354–365, 2011.

Expressiveness of STWs suffers from two limitations: 1) every node is visited exactly once, and 2) the nodes are visited in the fix left-to-right preorder traversal of the input tree. Consequently, STWs cannot express transformations that reorder the nodes of the input tree or make multiple copies of a part of the input document. STWs remain, however, very powerful and 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 [5,11].

Normal forms of transducers are typically obtained in two steps: output normalization followed by machine minimization. A natural way of output normalization is (re)arranging output words among the transitions rules so that the output is produced as soon as possible when reading the input, and thus transducers producing output in this fashion are called *earliest*. Our method subscribes to this approach but we note that it is a challenging direction that is not always feasible in the context of tree transformations. For instance, it fails for bottom-up tree-to-tree transducers, where *ad-hoc* solutions need to be employed [7].

We propose a natural normal form for STWs because on being earliest for STWs and define the corresponding class of *earliest* STWs (eSTWs) using easy to verify structural requirements. We present an effective procedure to convert an STW to an equivalent eSTW. This process is very challenging and requires novel tools on word languages. We point out that while this procedure works in time polynomial in the size of the output eSTW, we only know a doubly-exponential upper-bound and a single-exponential lower bound of the size of the output eSTW. This high complexity springs from the fact that the output language of an STW may be an arbitrary context-free language. We also show that minimization of earliest STWs is in PTIME thanks to a fundamental property: two equivalent eSTWs have rules of the same form and allow bisimulation. General STWs are unlikely to enjoy a similar property because their minimization is NP-complete.

Overall, we obtain an effective normalization procedure for STWs. Our results also offer an important step towards a better understanding of the same problem for dN2Ws because STWs capture a large class of top-down dN2Ws modulo the standard first-child next-sibling encoding and the conversion from one model to another can be done efficiently [16]. It is a significant result because there exist arguments suggesting that arbitrary dN2Ws are unlikely to have natural normal forms [1].

Organization. In Section 2 we present basic notions and introduce STWs and eSTWs. Section 3 introduces important tools on word languages and presents an STW to eSTW conversion algorithm. In Section 4 we deal with minimization of STWs and eSTWs. Section 5 summarizes our work and outlines future directions. Because of space restrictions we omit the proofs, which can be found in the full version at `http://hal.inria.fr/inria-00566291/en/`.

2 Sequential Top-Down Tree-to-Word Transducers

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 assume that every symbol has a unique arity i.e., $\Sigma^{(i)} \cap \Sigma^{(j)} = \varnothing$ for $i \neq j$. We use $f, g, \ldots$ to range over symbols of non negative arity and $a, b, \ldots$ to range over *constants* i.e., symbols of arity 0. We write $f^{(k)}$ to indicate that $f \in \Sigma^{(k)}$ if the arity of f is not known from the context. A *tree* is a ranked ordered term over Σ. We use $t, t_0, t_1, \ldots$ to range over trees. For instance, $t_0 = f(a, g(b))$ is a tree over $\Sigma = \{f^{(2)}, g^{(1)}, a^{(0)}, b^{(0)}\}$.

For a finite set Δ of symbols by Δ^* we denote the free monoid on Δ. We write $u \cdot v$ for the concatenation of two words u and v and ε for the empty word. We use $a, b, \ldots$ to range over Δ and $u, v, w, \ldots$ to range over Δ^*. For a word w by $|w|$ we denote its length. Given a word $u = u_p \cdot u_f \cdot u_s$, 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 nonempty set of words W, denoted $lcp(W)$, is the longest word u that is a prefix of every word in W. Analogously, we define the *longest common suffix* $lcs(W)$.

Definition 1. *A* deterministic sequential top-down tree-to-word transducer (STW) *is a tuple* $M = (\Sigma, \Delta, Q, init, \delta)$*, where* Σ *is a ranked alphabet of* input trees*,* Δ *is a finite alphabet of* output words*,* Q *is a finite set of states,* $init \in \Delta^* \cdot Q \cdot \Delta^*$ *is the* initial rule*,* 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 exactly* k *occurrences of elements from* Q*. By* STWs *we denote the class of deterministic sequential top-down tree-to-word transducers.*

In the sequel, if $u_0 \cdot q_0 \cdot u_1$ is the initial rule, then we call q_0 the initial state. Also, 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. The *size* of the STW M is the number of its states and the lengths of its rules, including the lengths of words used in the rules. The semantics of the STW M is defined with the help of auxiliary partial functions T_q (for $q \in Q$), recursively defined on the structure of trees as follows:

$$T_q(f(t_1, \ldots, t_k)) = \begin{cases} u_0 \cdot T_{q_1}(t_1) \cdot u_1 \cdot \ldots \cdot T_{q_k}(t_k) \cdot u_k, \\ \qquad\qquad \text{if } \delta(q, f) = u_0 \cdot q_1 \cdot u_1 \ldots \cdot q_k \cdot u_k, \\ \text{undefined}, \qquad \text{if } \delta(q, f) \text{ is undefined.} \end{cases}$$

The *transformation* T_M defined by M is a partial function mapping trees over Σ to words over Δ defined by $T_M(t) = u_0 \cdot T_{q_0}(t) \cdot u_1$, where $init = u_0 \cdot q_0 \cdot u_1$. Two transducers are *equivalent* iff they define the same transformation.

Example 1. We fix the input alphabet $\Sigma = \{f^{(2)}, g^{(1)}, a^{(0)}\}$ and the output alphabet $\Delta = \{a, b, c\}$. The STW M_1 has the initial rule q_0 and the following transition rules:

$$\delta(q_0, f) = q_1 \cdot ac \cdot q_1, \qquad \delta(q_1, g) = q_1 \cdot abc, \qquad \delta(q_1, a) = \varepsilon.$$

It defines the transformation $T_{M_1}(f(g^m(a), g^n(a))) = (abc)^m ac(abc)^n$, where $m, n \geq 0$, and T_{M_1} is undefined on all other input trees. The STW M_2 has the initial rule p_0 and these transition rules:

$$\delta(p_0, f) = p_1 \cdot p_3 \cdot ab, \quad \delta(p_1, g) = a \cdot p_2, \quad \delta(p_2, g) = ab \cdot p_3, \quad \delta(p_3, g) = p_3,$$
$$\delta(p_0, a) = ba, \quad \delta(p_1, a) = \varepsilon, \quad \delta(p_2, a) = \varepsilon, \quad \delta(p_3, a) = \varepsilon.$$

Now, $T_{M_2}(a) = ba$ and for $n \geq 0$, the result of $T_{M_2}(f(g^m(a), g^n(a))$ is ab for $m = 0$, aab for $m = 1$, and $aabab$ for $m \geq 2$; T_{M_2} is undefined for all other input trees. Note that p_3 is a *deleting* state: it does not produce any output but allows to check that the input tree belongs to the domain of the transducer. □

In the sequel, to simplify notation we assume every state belongs to exactly one transducer, and so T_q above is defined in unambiguous manner. We consider only trimmed STWs i.e., transducers where all states define a nonempty transformation and are accessible from the initial rule. Also, by dom_q we denote the set $dom(T_q)$, the domain of T_q i.e., the set of trees on which T_q is defined, and by L_q the range of T_q i.e., the set of words returned by T_q. For instance, $dom_{q_0} = \{f(g^m(a), g^n(a) \mid m, n \geq 0\}$ and $L_{q_0} = (abc)^* ac(abc)^*$. We observe that dom_q is a regular tree language and L_q is a context-free word language (CFL).

Next, we introduce the notion of being *earliest* that allows us to identify normal forms of transformations definable with STWs. It is a challenging task because the notion of being earliest needs to be carefully crafted so that every transducer can be made earliest. Take, for instance, the transformation *turn* that takes a tree over $\Sigma = \{a^{(1)}, b^{(1)}, \bot^{(0)}\}$ and returns the sequence of its labels in the reverse order e.g., $turn(a(b(b(\bot)))) = bba$. It is definable with a simple STW.

$$\delta(q_{turn}, a) = q_{turn} \cdot a, \qquad \delta(q_{turn}, b) = q_{turn} \cdot b, \qquad \delta(q_{turn}, \bot) = \varepsilon.$$

One way to view the transformation is a preorder traversal of the input tree that produces one output word upon entering the node and another word prior to leaving the node. When analyzing *turn* from this perspective, the earliest moment to produce any output is when the control reaches $\bot$, and in fact, the whole output can be produced at that point because all labels have been seen. This requires storing the label sequence in memory, which is beyond the capabilities of a finite state machine, and thus, *turn* cannot be captured with a transducer satisfying this notion of being earliest.

We propose a notion of being *earliest* that is also based on preorder traversal but with the difference that both output words are specified on entering the node and the output of a node is constructed right before leaving the node. Intuitively, we wish to *push up* all possible factors in the rules. Clearly, the STW above satisfies the condition. We remark that in some cases the output words in the rule can be placed in several positions, e.g. the rule $\delta(q_1, g) = q_1 \cdot abc$ in M_1 (Examples 1) can be replaced by $\delta(q_1, g) = abc \cdot q_1$ without changing the semantics of M_1. Consequently, we need an additional requirement that resolves this ambiguity: intuitively, we wish to *push left* the words in a rule as much as possible.

Definition 2. *An* STW $M = (\Sigma, \Delta, Q, init, \delta)$ *is* earliest (eSTW) *iff the following two conditions are satisfied:*

(E$_1$) $lcp(L_q) = \varepsilon$ *and* $lcs(L_q) = \varepsilon$ *for every state* q,
(E$_2$) $lcp(L_{q_0} \cdot u_1) = \varepsilon$ *for the initial rule* $u_0 \cdot q_0 \cdot u_1$ *and for every transition* $\delta(q, f) = u_0 \cdot q_1 \cdot \ldots \cdot q_k \cdot u_k$ *and* $1 \leq i \leq k$ *we have* $lcp(L_{q_i} \cdot u_i \cdot \ldots \cdot L_{q_k} \cdot u_k) = \varepsilon$.

Intuitively, the condition **(E$_1$)** ensures that no factor can be pushed up in the traversal and **(E$_2$)** ensures that no factor can be pushed left. We note that **(E$_1$)** and **(E$_2$)** can be efficiently checked in an STW because we need only to check that the results of *lcp* and *lcs* are ε. The main contribution of this paper follows.

Theorem 1. *For every* STW *there exists a unique minimal equivalent* eSTW.

The proof consists of an effective procedure that works in two stages: In the first stage we normalize the outputs, i.e. from the input STW we construct an equivalent eSTW, and in the second stage we minimize the obtained eSTW. The first stage is, however, quite complex as illustrated in the following example.

Example 2 (contd. Example 1). M_1 is not earliest because **(E$_1$)** is not satisfied at q_0: every word of $L_{q_0} = (abc)^* ac(abc)^*$ begins with a i.e., $lcp(L_{q_0}) = a$, and ends with c i.e., $lcs(L_{q_0}) = c$. Consequently, we need to *push up* these two symbols to the new initial rule $a \cdot q_0' \cdot c$, but we also need to retract them from the rule $\delta(q_0, f) = q_1 \cdot ac \cdot q_1$ producing a new state q_0' and new rules for this state. Essentially, we need to push the symbol a to the left through the first occurrence of q_1 and push the symbol c to the right through the second occurrence of q_1. Pushing symbols through states produces again new states with rules obtained by reorganizing the output words. Finally, we obtain

$$\delta'(q_0', f) = q_1' \cdot q_1'', \quad \delta'(q_1', g) = bca \cdot q_1', \quad \delta'(q_1'', g) = cab \cdot q_1'', \quad \delta'(q_1', a) = \delta'(q_1'', a) = \varepsilon.$$

M_2 is not earliest because **(E$_2$)** is not satisfied by $\delta(p_0, f) = p_1 \cdot p_3 \cdot ab$: every word produced by this rule starts with a. First, we push the word ab through the state p_3, and then we push the symbol a through the state p_1. Pushing through p_3 is easy because it is a deleting state and the rules do not change. Pushing through p_1 requires a recursive push through the states of the rules of p_1 and this process affects the rules of p_2. Finally, we obtain an eSTW with the initial rule p_0' and the transition rules

$$\begin{array}{llll} \delta'(p_0', f) = a \cdot p_1' \cdot b \cdot p_3', & \delta'(p_1', g) = a \cdot p_2', & \delta'(p_2', g) = ba \cdot p_3', & \delta'(p_3', g) = p_3', \\ \delta'(p_0', a) = ba, & \delta'(p_1', a) = \varepsilon, & \delta'(p_2', a) = \varepsilon, & \delta'(p_3', a) = \varepsilon. \quad \square \end{array}$$

3 Output Normalization

The first phase of normalization of an STW consists of constructing an equivalent eSTW, which involves changing the placement of the factors in the rules of the transducer and deals mainly with the output. Consequently, we begin with several notions and constructions inspired by the conditions **(E$_1$)** and **(E$_2$)** but set in a simpler setting of word languages. We consider only nonempty languages because in trimmed STWs the ranges of the states are always nonempty.

3.1 Reducing Languages

Enforcement of **($\mathbf{E_1}$)** corresponds to what we call constructing the *reduced decomposition* of a language. A nonempty language L is *reduced* iff $lcp(L) = \varepsilon$ and $lcs(L) = \varepsilon$. Note that the assumption that we work with a nonempty language is essential here. Now, take a nonempty language L, that is not necessarily reduced. We decompose it into its *reduced core* $Core(L)$ and two words $Left(L)$ and $Right(L)$ such that $Core(L)$ is reduced and

$$L = Left(L) \cdot Core(L) \cdot Right(L). \tag{1}$$

We observe that different decompositions are possible. For instance, $L = \{a, aba\}$ has two decompositions $L = a \cdot \{\varepsilon, ba\} \cdot \varepsilon$ and $L = \varepsilon \cdot \{\varepsilon, ab\} \cdot a$. We resolve the ambiguity by choosing the former decomposition because it is consistent with **($\mathbf{E_1}$)** and **($\mathbf{E_2}$)** which indicate to *push to the left*. Formally, $Left(L) = lcp(L)$ and $Right(L) = lcs(L')$, where $L = Left(L) \cdot L'$. The reduced core $Core(L)$ is obtained from (1). As an example, the reduced decomposition of $L_{q_0} = (abc)^*ac(abc)^*$ from Example 1 is $Left(L_{q_0}) = a$, $Right(L_{q_0}) = c$, and $Core(L_{q_0}) = (bca)^*(cba)^*$.

3.2 Pushing Words through Languages

In this subsection, we work with *nonempty* and *reduced* languages only. Condition **($\mathbf{E_2}$)** introduces the problem that we call pushing words through languages. To illustrate it, suppose we have a language $L = \{\varepsilon, a, aa, aaab\}$ and a word $w = aab$, which together give $L \cdot w = \{aab, aaab, aaaab, aaabaab\}$. The goal is to find the longest prefix v of w such that $L \cdot w = v \cdot L' \cdot u$, where $w = v \cdot u$ and L' is some derivative of L. Intuitively speaking, we wish to push (a part of) the word w forward i.e., from right to left, through the language L. In the example above, the solution is $v = aa$, $L' = \{\varepsilon, a, aa, abaa\}$, and $u = b$ (note that L' is different from L). In this section, we show that this process is always feasible and for CFLs it is constructive.

The result of pushing a word w through a language L will consist of three words: $push(L, w)$ the longest part of w that can be pushed through L, $rest(L, w)$ the part that cannot be pushed through, and $offset(L, w)$ a special word that allows to identify the corresponding derivative of L. There are three classes of languages that need to be considered, which we present next together with an outline of how the pushing is done.

The first class contains only the *trivial* language $L = \{\varepsilon\}$ e.g., the range of the state p_3 of M_2 in Example 1. This language allows every word to be pushed through and it never changes in the process. For instance, if $w_0 = ab$, then $push(L_{p_3}, w_0) = ab$, $rest(L_{p_3}, w_0) = \varepsilon$, and $offset(L_{p_3}, w_0) = \varepsilon$.

The second class consists of non-trivial periodic languages, essentially languages contained in the Kleene closure of some period word. An example is $L_{q_1} = (abc)^* = \{\varepsilon, abc, abcabc, \ldots\}$ whose period is abc. Periodic languages allow to push multiplicities of the period and then some prefix of the period e.g., if we take $w_1 = abcabcaba$, then $push(L_{q_1}, w_1) = abcabcab$ and $rest(L_{q_1}, w_1) = a$. The offset here is the corresponding prefix of the period: $offset(L_{q_1}, w_1) = ab$.

The third class contains all remaining languages i.e., non-trivial non-periodic languages. Interestingly, we show that for a language in this class there exists a word that is the longest word that can be pushed fully through the language, and furthermore, every other word that can be pushed through is a prefix of this word. For instance, for $L_{p_1} = \{\varepsilon, a, aab\}$ from Example 1, aa is the longest word that can be pushed through. If we take $w_2 = ab$, then we get $push(L_{p_1}, w_2) = a$ and $rest(L_{p_1}, w_2) = b$. Here, the offset is the prefix of aa that has been already pushed through: $offset(L_{p_1}, w_2) = a$. Note that this class also contains the languages that do not permit any pushing through e.g., $L_{p_0} = \{ba, ab, aab\}$ does not allow pushing through because it contains two words that start with a different symbol.

We now define formally the pushing process. First, for $L \subseteq \Delta^*$ we define the set of words that can be pushed fully through L:

$$Shovel(L) = \{w \in \Delta^* \mid w \text{ is a common prefix of } L \cdot w\}.$$

For instance, $Shovel(L_{p_1}) = \{\varepsilon, a, aa\}$ and $Shovel(L_{q_0}) = (abc)^* \cdot \{\varepsilon, a, ab\}$. We note that $Shovel(\{\varepsilon\}) = \Delta^*$ and $Shovel(L)$ always contains at least one element ε because L is assumed to be nonempty. Also, as we prove in appendix, $Shovel(L)$ is prefix-closed and totally ordered by the prefix relation.

Next, we define periodic languages (cf. [12]). A language $L \subseteq \Delta^*$ is *periodic* iff there exists a nonempty word $v \in \Delta^*$, called a *period* of L, such that $L \subseteq v^*$. A word w is *primitive* if there is no v and $n \geq 0$ such that $w = v^n$. Recall from [12] that every non-trivial periodic language L has a unique primitive period, which we denote $Period(L)$. For instance, the language $\{\varepsilon, abab, abababab\}$ is periodic and its primitive period is ab; $abab$ is also its period but not primitive. In the sequel, by $Prefix(w)$ we denote the set of prefixes of the word w.

Proposition 1. *Given a reduced and non-trivial language L, $Shovel(L)$ is infinite iff L is periodic. Furthermore, if L is periodic then $Shovel(L) = Period(L)^* \cdot Prefix(Period(L))$.*

This result and the observations beforehand lead to three relevant cases in the characterisation of $Shovel(L)$ for a language L.

0^o $L = \{\varepsilon\}$ (trivial language), and then $Shovel(L) = \Delta^*$,
1^o L is periodic, $L \neq \{\varepsilon\}$, and then $Shovel(L) = Period(L)^* \cdot Prefix(Period(L))$.
2^o L is non-periodic, and $Shovel(L) = Prefix(v)$ for some $v \in Shovel(L)$.

Now, suppose we wish to *push* a word $w \in \Delta^*$ *through* a language $L \subseteq \Delta^*$ and let $s = \max_{\leq_{prefix}}(Prefix(w) \cap Shovel(L))$ and $w = s \cdot r$. We define $push(L, w)$, $rest(L, w)$, and $offset(L, w)$ depending on the class L belongs to:

0^o $L = \{\varepsilon\}$: $push(L, w) = w$, $rest(L, w) = \varepsilon$, and $offset(L, w) = \varepsilon$.
1^o L is non-trivial and periodic: $s = Period(L)^k \cdot o$ for some (maximal) proper prefix o of $Period(L)$, and we assign $push(L, w) = s$, $rest(L, w) = r$, and $offset(L, w) = o$.
2^o L is non-periodic: $push(L, w) = s$, $rest(L, w) = r$, and $offset(L, w) = s$.

Offsets play a central role in the output normalization procedure, which is feasible thanks to the following result.

Proposition 2. *The set $\{offset(L, w) \mid w \in \Delta^*\}$ is finite for any reduced L.*

3.3 Pushing Words Backwards

Until now, we have considered the problem of pushing a word through a language from right to left. However, in Example 1 if we consider the second occurrence of q_1 in the rule $\delta(q_0, f) = q_1 \cdot ac \cdot q_1$, we realize that pushing words in the opposite direction needs to be investigated as well. These two processes are dual but before showing in what way, we present a natural extension of the free monoid Δ^* to a pregroup (or groupoid) $\mathbb{G}_\Delta$. It allows to handle pushing in two directions in a unified manner and simplifies the output normalization algorithm.

A *pregroup of words over* Δ is the set $\mathbb{G}_\Delta = \Delta^* \cup \{w^{-1} \mid w \in \Delta^+\}$, where w^{-1} is a term representing the inverse of an nonempty word w. This set comes with two operators, a unary inverse operator: $(w)^{-1} = w^{-1}$, $\varepsilon^{-1} = \varepsilon$, and $(w^{-1})^{-1} = w$ for $w \in \Delta^*$, and a partial extension of the standard concatenation that satisfies the following equations (complete definition in appendix): $w^{-1} \cdot w = \varepsilon$ and $w \cdot w^{-1} = \varepsilon$ for $w \in \Delta^*$, and $v^{-1} \cdot u^{-1} = (uv)^{-1}$ for $u, v \in \Delta^*$. We note that some expressions need to be evaluated diligently e.g., $ab \cdot (cb)^{-1} \cdot cd = ab \cdot b^{-1} \cdot c^{-1} \cdot cd = ad$, while some are undefined e.g., $ab \cdot a^{-1}$. In the sequel, we use $w, u, v, \ldots$ to range over Δ^* only and $z, z_1, \ldots$ to range over elements of $\mathbb{G}_\Delta$.

Now, we come back to pushing a word w backwards through L, which consists of finding $u \cdot v = w$ and L' such that $w \cdot L = u \cdot L' \cdot v$. We view this process as pushing the inverse w^{-1} through L i.e., we wish to find $u \cdot v = w$ such that $L \cdot w^{-1} = v^{-1} \cdot L' \cdot u^{-1}$ because then $L \cdot v^{-1} = v^{-1} \cdot L'$, and consequently, $w \cdot L = (u \cdot v) \cdot (v^{-1} \cdot L' \cdot v) = u \cdot L' \cdot v$.

But to define pushing backwards more properly we use another perspective based on the standard reverse operation of a word e.g., $(abc)^{\mathrm{rev}} = cba$. Namely, pushing w backwards through L is essentially pushing w^{rev} through L^{rev} because $(w \cdot L)^{\mathrm{rev}} = L^{\mathrm{rev}} \cdot w^{\mathrm{rev}}$ and if $L^{\mathrm{rev}} \cdot w^{\mathrm{rev}} = v_0 \cdot L_0 \cdot u_0$, then $w \cdot L = u_0^{\mathrm{rev}} \cdot L_0^{\mathrm{rev}} \cdot v_0^{\mathrm{rev}}$. Thus $\mathit{push}(L, w^{-1}) = (\mathit{push}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}$, $\mathit{rest}(L, w^{-1}) = (\mathit{rest}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}$, and $\mathit{offset}(L, w^{-1}) = (\mathit{offset}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}$.

Now, the main condition of pushing words through languages is: for every L and $z \in \mathbb{G}_\Delta$ we have $L \cdot z = \mathit{push}(L, z) \cdot (\mathit{offset}(L, z)^{-1} \cdot L \cdot \mathit{offset}(L, z)) \cdot \mathit{rest}(L, z)$. Because the output normalization procedure works on STWs and not languages, to prove its correctness we need a stronger statement treating independently every word of the language.

Proposition 3. *Given a reduced and nonempty language* $L \subseteq \Delta^*$ *and* $z \in \mathbb{G}_\Delta$, *for any word* $u \in L$

$$u \cdot z = \mathit{push}(L, z) \cdot (\mathit{offset}(L, z)^{-1} \cdot u \cdot \mathit{offset}(L, z)) \cdot \mathit{rest}(L, z).$$

3.4 Output Normalization Algorithm

We fix an STW $M = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ and introduce the following macros:

$$\hat{L}_q = \mathit{Core}(L_q), \quad \mathit{Left}(q) = \mathit{Left}(L_q), \quad \mathit{Right}(q) = \mathit{Right}(L_q),$$
$$\mathit{push}(q, z) = \mathit{push}(\hat{L}_q, z), \quad \mathit{offset}(q, z) = \mathit{offset}(\hat{L}_q, z), \quad \mathit{rest}(q, z) = \mathit{rest}(\hat{L}_q, z).$$

Also, let $\mathit{Offsets}(q) = \{\mathit{offset}(q, z) \mid z \in \mathbb{G}_\Delta\}$ and note that by Proposition 2 it is finite. The constructed STW $M' = (\Sigma, \Delta, Q', \mathit{init}', \delta')$ has the following states

$$Q' = \{\langle q, w\rangle \mid q \in Q,\ w \in \mathit{Offsets}(q)\}.$$

Our construction ensures that $T_M = T_{M'}$ and for every $q \in Q$, every $z \in \mathit{Offsets}(q)$, and every $t \in \mathit{dom}_q$

$$T_{\langle q,z\rangle}(t) = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot T_q(t) \cdot \mathit{Right}(q)^{-1} \cdot z$$

If $\mathit{init} = u_0 \cdot q_0 \cdot u_1$, then $\mathit{init}' = u_0' \cdot q_0' \cdot u_1'$, where u_0', u_1', and q_0' are calculated as follows:

1: $v := \mathit{Right}(q_0) \cdot u_1$
2: $q_0' := \langle q_0, \mathit{offset}(q_0, v)\rangle$
3: $u_0' := u_0 \cdot \mathit{Left}(q_0) \cdot \mathit{push}(q_0, v)$
4: $u_1' := \mathit{rest}(q_0, v)$

For a transition rule $\delta(p, f) = u_0 \cdot p_1 \cdot u_1 \cdot \ldots \cdot u_{k-1} \cdot p_k \cdot u_k$ and any $z \in \mathit{Offsets}(p)$ we introduce a rule $\delta'(\langle p, z\rangle, f) = u_0' \cdot p_1' \cdot u_1' \cdot \ldots \cdot u_{k-1}' \cdot p_k' \cdot u_k'$, where $u_0', \ldots, u_k'$ and $p_1', \ldots p_k'$ are calculated as follows:

1: $z_k := \mathit{Right}(p_k) \cdot u_k \cdot \mathit{Right}(p)^{-1} \cdot z$
2: **for** $i := k, \ldots, 1$ **do**
3: $\quad u_i' := \mathit{rest}(p_i, z_i)$
4: $\quad p_i' := \langle p_i, \mathit{offset}(p_i, z_i)\rangle$
5: $\quad z_{i-1} := \mathit{Right}(p_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(p_i) \cdot \mathit{push}(p_i, z_i)$
6: $u_0' := z^{-1} \cdot \mathit{Left}(p)^{-1} \cdot z_0$

where (for convenience of the presentation) we let $\mathit{Right}(p_0) = \varepsilon$. We remark that not all states in Q' are reachable from the initial rule and in fact the conversion procedure can identify the reachable states *on the fly*. This observation is the basis of a conversion algorithm that is polynomial in the size of the output.

Example 3. We normalize the STW M_1 from Example 1. The initial rule q_0 becomes $a \cdot \langle q_0, \varepsilon\rangle \cdot c$ with $\mathit{Left}(q_0) = a$ and $\mathit{Right}(q_0) = c$ being pushed up from q_0 but with nothing pushed through q_0. The construction of the state $\langle q_0, \varepsilon\rangle$ triggers the normalization algorithm for the rule $\delta(q_0, f) = q_1 \cdot ac \cdot q_1$ with $\mathit{Left}(q_0) = a$ and $\mathit{Right}(q_0) = c$ to be retracted from left and right side resp. (and nothing pushed through since $z = \varepsilon$). This process can be viewed as a taking the left hand side of the original rule with the inverses of retracted words $a^{-1} \cdot q_1 \cdot ac \cdot q_1 \cdot c^{-1}$ and pushing words forward as much as possible, which gives $a^{-1} \cdot q_1 \cdot ac \cdot c^{-1} \cdot \langle q_1, c^{-1}\rangle$ and then $a^{-1} \cdot a \cdot \langle q_1, a\rangle \cdot \langle q_1, c^{-1}\rangle$. This gives $\delta'(\langle q_0, \varepsilon\rangle, f) = \langle q_1, a\rangle \cdot \langle q_1, c^{-1}\rangle$. Note that while $\mathit{Offsets}(q_1) = \{(bc)^{-1}, c^{-1}, \varepsilon, a, ab\}$, only two states are constructed.

Next, we need to construct rules for the new state $\langle q_1, a\rangle$ with $z = a$ and $\mathit{Left}(q_1) = \mathit{Right}(q_1) = \varepsilon$. We start with the rule $\delta(q_1, a) = \varepsilon$ and to its left hand side we add a^{-1} at the beginning and a at its end: $a^{-1} \cdot \varepsilon \cdot a = \varepsilon$, which yields the rule $\delta'(\langle q_1, a\rangle, a) = \varepsilon$. Now, for the rule $\delta(q_1, g) = q_1 \cdot abc$ we obtain the expression $a^{-1} \cdot q_1 \cdot abca$. Recall that $L_{q_1} = (abc)^*$ is a periodic language, and so

$push(q_1, abca) = abca$, $rest(q_1, abca) = \varepsilon$, and $offset(q_1, abca) = a$. Consequently, we obtain the rule $\delta'(\langle q_1, a\rangle, g) = bca \cdot \langle q_1, a\rangle$. Here, it is essential to use the offsets to avoid introducing a redundant state $\langle q_1, abca\rangle$ and entering an infinite loop. Similarly, we obtain: $\delta'(\langle q_1, c^{-1}\rangle, g) = cab \cdot \langle q_1, c^{-1}\rangle$ and $\delta'(\langle q_1, c^{-1}\rangle, a) = \varepsilon$. □

Theorem 2. *For an* STW M *let* M' *be the* STW *obtained with the method described above. Then,* M' *is equivalent to* M *and satisfies* $\mathbf{(E_1)}$ *and* $\mathbf{(E_2)}$*. Furthermore,* M' *can be constructed in time polynomial in the size* M'*, which is at most doubly-exponential in the size of* M*.*

Because of space limitations, the details on complexity have been omitted and can be found in the full version available online.

3.5 Exponential Lower Bound

First, we show that the size of a rule may increase exponentially.

Example 4. For $n \geq 0$ define an STW M_n over the input alphabet $\Sigma = \{f^{(2)}, a^{(0)}\}$ with the initial rule q_0, and these transition rules (with $0 \leq i < n$):

$$\delta(q_i, f) = q_{i+1} \cdot q_{i+1}, \qquad \delta(q_n, a) = a.$$

The transformation defined by M_n maps a perfect binary tree of height n to a string a^{2^n}. M_n is not earliest. To make it earliest we need to replace the initial rule by $a^{2^n} \cdot q_0(x_0)$ and the last transition rule by $\delta(q_n, a) = \varepsilon$. □

The next example shows that also the number of states may become exponential.

Example 5. For $n \geq 0$ and take the STW N_n with $\Sigma = \{g_1^{(1)}, g_0^{(1)}, a_1^{(0)}, a_0^{(0)}\}$, the initial rule q_0, and these transition rules (with $0 \leq i < n$):

$$\delta(q_i, g_0) = q_{i+1}, \qquad \delta(q_n, a_0) = \varepsilon,$$
$$\delta(q_i, g_1) = q_{i+1} \cdot a^{2^i}, \qquad \delta(q_n, a_1) = a^{2^n} \cdot \#.$$

While the size of this transducer is exponential in n, one can easily compress the exponential factors 2^{2^i} and obtain an STW of size linear in n (cf. Example 4). M_n satisfies $\mathbf{(E_1)}$ but it violates $\mathbf{(E_2)}$, and defines the following transformation.

$$T_{N_n} = \{(g_{b_0}(g_{b_1}(\ldots g_{b_{n-1}}(a_0)\ldots)), a^{\mathbf{b}}) \mid \mathbf{b} = (b_{n-1}, \ldots, b_0)_2\} \cup$$
$$\{(g_{b_0}(g_{b_1}(\ldots g_{b_{n-1}}(a_1)\ldots)), a^{2^n} \cdot \# \cdot a^{\mathbf{b}}) \mid \mathbf{b} = (b_{n-1}, \ldots, b_0)_2\},$$

where $(b_{n-1}, \ldots, b_0)_2 = \sum_i b_i * 2^i$. The normalized version N'_n has the initial rule $\langle q_0, \varepsilon\rangle$ and these transition rules:

$$\delta'(\langle q_i, a^j\rangle, g_0) = \langle q_{i+1}, a^j\rangle, \qquad \delta'(\langle q_n, a^k\rangle, a_0) = \varepsilon,$$
$$\delta'(\langle q_i, a^j\rangle, g_1) = a^{2^i} \cdot \langle q_{i+1}, a^{j+2^i}\rangle, \qquad \delta'(\langle q_n, a^k\rangle, a_1) = a^{2^n - k} \# a^k,$$

where $0 \leq i < n$, $0 \leq j < 2^i$, and $0 \leq k < 2^n$. We also remark that N'_n is the minimal eSTW that recognises T_{N_n}. □

4 Minimization

In this section we investigate the problem of minimizing the size of a transducer. Minimization of eSTWs is simple and relies on testing the equivalence of eSTWs known to be in PTIME [16]. For an eSTW M the minimization procedure constructs a binary equivalence relation $\equiv_M$ on states such that $q \equiv_M q'$ iff $T_q = T_{q'}$. The result of minimization is the quotient transducer $M/_{\equiv_M}$ obtained by choosing in every equivalence class C of $\equiv_M$ exactly one representative state $q \in C$, and then replacing in rules of M every state of C by q.

To show that the obtained eSTW is minimal among all eSTWs defining the same transformation, we use an auxiliary result stating that all eSTWs defining the same transformation use rules with the same distribution of the output words and allow bisimulation.

A *labeled path* is a word over $\bigcup_{k>0} \Sigma^{(k)} \times \{1, \ldots, k\}$, which identifies a node in a tree together with the labels of its ancestors: ε is the root node and if a node π is labeled with f, then $\pi \cdot (f, i)$ is its i-th child. By $paths(t)$ we denote the set of labeled paths of a tree t. For instance, for $t_0 = f(a, g(b))$ we get $paths(t_0) = \{\varepsilon, (f,1), (f,2), (f,2)\cdot(g,1)\}$. We extend the transition function δ to identify the state reached at a path π: $\delta(q, \varepsilon) = q$ and $\delta(q, \pi \cdot (f, i)) = q_i$, where $\delta(q, \pi) = q'$ and $\delta(q', f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot q_k \cdot u_k$. Now, the lemma of interest.

Lemma 1. *Take two* eSTWs $M = (\Sigma, \Delta, Q, init, \delta)$ *and* $M' = (\Sigma, \Delta, Q', init', \delta')$ *defining the same transformation* $T = T_M = T_{M'}$ *and let* $init = u_0 \cdot q_0 \cdot u_1$ *and* $init' = u'_0 \cdot q'_0 \cdot u'_1$. *Then,* $u_0 = u'_0$ *and* $u_1 = u'_1$, *and for every* $\pi \in paths(dom(T))$, *we let* $q = \delta(q_0, \pi)$ *and* $q' = \delta'(q'_0, \pi)$, *and we have*

1. $T_q = T_{q'}$,
2. $\delta(q, f)$ *is defined if and only if* $\delta'(q', f)$ *is, for every* $f \in \Sigma$, *and*
3. *if* $\delta(q, f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot q_k \cdot u_k$ *and* $\delta'(q', f) = u'_0 \cdot q'_1 \cdot u'_1 \cdot \ldots \cdot q'_k \cdot u'_k$, *then* $u_i = u'_i$ *for* $0 \leq i \leq k$.

The proof is inductive and relies on properties **($\mathbf{E_1}$)** and **($\mathbf{E_2}$)**, and the determinism of the transducers. We show the correctness of our minimization algorithm by observing that it produces an eSTW whose size is smaller than the input one, and Lemma 1 essentially states that the result of minimization of two equivalent transducers is the same transducer (modulo state renaming). This argument also proves Theorem 1. We also point out that Lemma 1 (with $M = M'$) allows to devise a simpler and more efficient minimization algorithm along the lines of the standard DFA minimization algorithm [10].

Theorem 3. *Minimization of* eSTWs *is in* PTIME.

In STWs the output words may be arbitrarily distributed among the rules, which is the main pitfall of minimizing general STWs. This difficulty is unlikely to be overcome as suggested by the following result.

Theorem 4. *Minimization of* STWs *i.e., deciding whether for an* STW M *and* $k \geq 0$ *there exists an equivalent* STW M' *of size at most* k, *is* NP*-complete.*

5 Conclusions and Future Work

We have presented an effective normalization procedure for STWs, a subclass of top-down tree-to-word transducers closely related to a large subclass of nested-word to word transducers. One natural continuation of this work is find whether it can be extended to a Myhill-Nerode theorem for STWs, and then, to a polynomial learning algorithm. Also, the question of exact complexity of the normalization remains open. Finally, the model of STWs can be generalized to allow arbitrary non-sequential rules and multiple passes over the input tree.

References

1. Alur, R., Kumar, V., Madhusudan, P., Viswanathan, M.: Congruences for visibly pushdown languages. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1102–1114. Springer, Heidelberg (2005)
2. Berstel, J., Boasson, L.: Transductions and context-free languages. Teubner Studienbucher (1979)
3. Choffru, C.: Contribution à l'étude de quelques familles remarquables de fonctions rationnelles. PhD thesis, Université de Paris VII (1978)
4. Choffrut, C.: Minimizing subsequential transducers: a survey. Theoretical Computer Science 292(1), 131–143 (2003)
5. Engelfriet, J., Maneth, S., Seidl, H.: Deciding equivalence of top-down XML transformations in polynomial time. Journal of Computer and System Science 75(5), 271–286 (2009)
6. Filiot, E., Raskin, J.F., Reynier, P.A., Servais, F., Talbot, J.M.: Properties of visibly pushdown transducers. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 355–367. Springer, Heidelberg (2010)
7. Friese, S., Seidl, H., Maneth, S.: Minimization of deterministic bottom-up tree transducers. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 185–196. Springer, Heidelberg (2010)
8. Griffiths, T.V.: The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. Journal of the ACM 15(3), 409–413 (1968)
9. Gurari, E.M.: The equivalence problem for deterministic two-way sequential transducers is decidable. SIAM Journal on Computing 11(3), 448–452 (1982)
10. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley, Reading (2001)
11. Lemay, A., Maneth, S., Niehren, J.: A learning algorithm for top-down xml transformations. In: ACM Symposium on Principles of Database Systems (PODS), pp. 285–296 (2010)
12. Lothaire, M. (ed.): Combinatorics on Words, 2nd edn. Cambridge Mathematical Library. Cambridge University Press, Cambridge (1997)
13. Maneth, S.: Models of Tree Translation. PhD thesis, Leiden University (2003)
14. Martens, W., Neven, F., Gyssens, M.: Typechecking top-down XML transformations: Fixed input or output schemas. Information and Computation 206(7), 806–827 (2008)
15. Raskin, J.-F., Servais, F.: Visibly pushdown transducers. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 386–397. Springer, Heidelberg (2008)
16. Staworko, S., Laurence, G., Lemay, A., Niehren, J.: Equivalence of deterministic nested word to word transducers. In: Kutyłowski, M., Charatonik, W., Gębala, M. (eds.) FCT 2009. LNCS, vol. 5699, pp. 310–322. Springer, Heidelberg (2009)