Normal Form on Linear Tree-to-word Transducers

Adrien Boiret

Normal Form on Linear Tree-to-word Transducers

Adrien Boiret[1,2]

[1] University Lille 1, France
[2] Links (Inria Lille & CRIStAL, UMR CNRS 9189), France

Abstract. We study a subclass of tree-to-word transducers: linear tree-to-word transducers, that cannot use several copies of the input. We aim to study the equivalence problem on this class, by using minimization and normalization techniques. We identify a Myhill-Nerode characterization. It provides a minimal normal form on our class, computable in Exptime. This paper extends an already existing result on tree-to-word transducers without copy or reordering (sequential tree-to-word transducers), by accounting for all the possible reorderings in the output.

1 Introduction

Transducers and their properties have long been studied in various domains of computer sciences. The views on transducers that motivate this paper's field of research are mostly the result of the intersection of two approaches.

Language theory sees transducers as the natural extension of automata, with an output. This view extends almost as far back as the study of regular languages, and developed techniques to solve classical problems such as equivalence, typechecking, or even learning problems (e.g. [11, 10, 3]) on increasingly wide classes of transducers.

Functional programming sees transducers as a formal representation of some programs. In order to study languages such as Xslt, XQuery, or XProc, used to transform Xml trees, classes of transducers that acted more and more like functional programs were designed and studied. For example, deterministic top-down tree transducers can be seen as a functional program that transform trees from the root to the leaves, with finite memory. Different classes extend the reach of transducers to encompass more of the functionalities of programming languages.

Concatenation in the output, notably, plays an important role in the way Xslt produces its outputs. Classes like macro-tree transducers [6], tree-to-word transducers, or even word-to-word transducers with copies in the output [1] allow such concatenation, but as this functionality appears to be difficult to combine with the classical techniques of language theory, this is to the cost of very few results carrying to these classes.

Tree-to-word transducers and Macro-tree transducers are of particular relevance, as they allow concatenation in their output, and are at the current

frontier between the language theory approach of transducers and the approach of transducers seen as functional programs.

Many problems are left open in these classes. Notably, in the general case for Macro-tree transducers, the decidability equivalence is a famous long-standing question, that has yet to be resolved. However, some pre-existing results exist for fragments of these classes.

Equivalence for the subclass of linear size increase macro-tree transducers [4] is proven to be decidable. It comes from a *logic* characterization, as if we bound the number of times a transducer can copy the same subtree in the output, then we limit the expressivity of macro-tree transducers into MSO-definable translations, where equivalence is decidable in non-elementary complexity [3].

Equivalence for all tree-to-word transducers has recently been proven to be decidable in randomized polynomial time [12]. Note that this result uses neither classic logic methods nor the classic transducer methods, and does not provide a characterization or Myhill-Nerode theorem.

Equivalence is Ptime for sequential tree-to-word transducers [7], that prevents copying in the output and forces subtrees to produce following the order of the input. Furthermore, using a *Myhill-Nerode* characterization, a normal form computable in Exptime is shown to exist. This normal form was later proven to be learnable in Ptime [8].

In this paper , we aim to study the linear tree-to-word transducers (or Ltws), a restriction of deterministic tree-to-word transducers that forbids copying in the output, but allows the image of subtrees to be flipped in any order. This is a more general class than sequential tree-to-word transducers, but still less descriptive than general tree-to-word transductions. In this class, we show the existence of a normal form, computable in Exptime.

Note that even if equivalence is already known to be decidable in a reasonable complexity, finding a normal form is of general interest in and of itself. For example, in [11, 9, 8], normal forms on transducers defined using a Myhill-Nerode theorem are used to obtain a learning algorithm.

To define a normal form on Ltws, we start by the methods used for sequential tree-to-words transducers (Stws) in [7]. We consider the notion of *earliest* Stws, which normalizes the output production. We can extend this notion to Ltws and study only earliest Ltws without any loss of expressivity.

In [7], this is enough to obtain a Myhill-Nerode characterization. However, by adding the possibility to flip subtree images to Ltws, we created another way for equivalent transducers to differ. The challenge presented by the extension of the methods of [7] becomes to resolve this new degree of freedom, in order to obtain a good normal form with a Myhill-Nerode characterization.

Outline. After introducing basic notions on words and trees, we will present our class of linear tree-to-word transducers in Section 2. Then in Section 3 we will extend the notion of *earliest* production in [7] to the linear case, and find out that we can also extend the algorithm that takes a transducer and compute and equivalent earliest one. However, this is no longer sufficient, as transducers can now also differ in the order they produce their subtrees' output in. Section 4

will detail exactly how two earliest transducers can still differ, by categorizing all possible flips. Finally, Section 5 will compile these results into a Myhill-Nerode theorem. This will allow us to establish a normal form, computable in EXPTIME. We will conclude by a brief recap of the result, and propose several possible next steps for this line of research.

2 Preliminaries

Words and Trees

We begin by fixing notations on standard notions over words and ranked trees.

Words. For a finite set of symbols Δ, we denote by Δ^* 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 a set of words L, we denote $lcp(L)$ the longest word u that is a prefix of every word in L, or *largest common prefix*. Also, $lcs(L)$ is the largest 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 $w \cdot v^{-1} = u$.

Ranked Trees. 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. Every symbol has a unique arity. A *tree* is a ranked ordered term over Σ. For example, $t = f(a, g(b))$ is a tree over Σ if $f \in \Sigma^{(2)}$, $g \in \Sigma^{(1)}$, $a, b \in \Sigma^{(0)}$. The set of all trees on Σ is $\mathcal{T}$.

Linear Tree-to-Word Transducers

We define linear tree-to-word transducer, that define a function from $\mathcal{T}$ to Δ^*.

Definition 1. *A linear tree-to-word transducer (*LTW*) is a tuple* $M = \{\Sigma, \Delta, Q, ax, rul\}$ *where*

- Σ *is a tree alphabet,*
- Δ *is a finite word alphabet of output symbols,*
- Q *is a finite set of states,*
- ax *is a axiom of form* $u_0 q u_1$*, where* $u_0, u_1 \in \Delta^*$ *and* $q \in Q$*,*
- rul *is a set of rules of the form*
$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$
where $q, q_1, \dots, q_n \in Q$*,* $f \in \Sigma$ *of rank* n *and* $u_0 \dots u_n \in \Delta^*$*;* σ *is a permutation on* $\{1, \dots, n\}$*. There is at most one rule per pair* q, f*.*

We define recursively the function $[\![M]\!]_q$ *of a state* q*.* $[\![M]\!]_q(f(t_1 ... t_n))$ *is*

- $u_0 [\![M]\!]_{q_1}(t_{\sigma(1)}) u_1 \dots [\![M]\!]_{q_n}(t_{\sigma(n)}) u_n$*,*
if $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n \in \delta$
- *undefined, if there is no rule for* q, f *in* δ*.*

The function $[\![M]\!]$ *of a transducer* M *with axiom* $u_0 q u_1$ *is defined as* $[\![M]\!](s) = u_0 [\![M]\!]_q(s) u_1$*.*

Note that to get the definition of STWs as made in [7], we just have to impose that in every rule, σ is the identity.

Example 2. Consider the function $[\![M]\!] : t \mapsto 0^{|t|}$, that counts the number of nodes in t and writes a 0 in the output for each of them. Our LTW has only one state q, and its axiom is $ax = q$

$$q(f(x_1, x_2)) \to 0 \cdot q(x_1) \cdot q(x_2)$$
$$q(a) \to 0, \quad q(b) \to 0$$

The image of $f(a, b)$ is $[\![M]\!](f(a, b)) = [\![M]\!]_q(f(a, b))$, using the axiom. Then we use the first rule to get $0 \cdot [\![M]\!]_q(a) \cdot [\![M]\!]_q(b)$, and finally, $0 \cdot 0 \cdot 0$

We denote with $dom([\![M]\!])$ the domain of a transducer M, i.e. all trees such that $[\![M]\!](t)$ is defined. Similarly, $dom([\![M]\!]_q)$ is the domain of state q.

We define accessibility between states as the transitive closure of appearance in a rule. This means q is accessible from itself, and if there is a rule $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$, and q accessible from q', then all states q_i, $1 \leq i \leq n$, are accessible from q'.

We note L_q the set of all productions of q: $L_q = \{[\![M]\!]_q(t) | t \in dom([\![M]\!]_q)\}$. We call a state *periodic* of period $w \in \Delta^*$ if $L_q \subseteq w^*$.

We start the normalization process with a natural notion of trimmed LTWs.

Definition 3. *A* LTW *is* trimmed *if its axiom is $u_0 q_0 v_0$, and every state q is accessible from q_0 and of non-empty domain.*

Note that all LTWs can be made trimmed by deleting all their useless states.

Lemma 4. *For M a* LTW*, one can compute an equivalent trimmed* LTW *in linear time.*

3 Earliest Linear Transducers

It is possible for different LTWs to encode the same transformation. To reach a normal form , we start by requiring our LTWs to produce their output "as soon as possible". This method is common for transducers [2, 5], and has been adapted to sequential tree-to-word transducers in [7]. In this case, the way an output word is produced by a tree-to-word can be "early" in two fashions: it can be produced sooner in the input rather than later, or it can output letters on the left of a rule rather than on the right. We take the natural extension of this definition for LTWs and find we can reuse the results and algorithms of [7].

Example 5. Consider our previous example (Ex. 2). The function $[\![M]\!] : t \mapsto 0^{|t|}$, Our transducer has only one state q, and its axiom is $ax = q$

$$q(f(x_1, x_2)) \to 0 \cdot q(x_1) \cdot q(x_2)$$
$$q(a) \to 0, \quad q(b) \to 0$$

Since all productions of q start with a 0, this LTW does not produce as up in the input as possible. To change this, we form a new state q' that produces one 0

less than q. By removing the 0 at the beginning of each rule of q, and replacing each call $q(x_i)$ by $0q'(x_i)$, we get a new equivalent Ltw M' of axiom $ax' = 0 \cdot q'$

$$q'(f(x_1, x_2)) \to 0 \cdot q'(x_1) \cdot 0 \cdot q'(x_2)$$
$$q'(a) \to \varepsilon \;\; q'(b) \to \varepsilon$$

Example 6. Consider our previous example (Ex. 5). We could replace the first rule by $q'(f(x_1, x_2)) \to 0 \cdot 0 \cdot q'(x_1) \cdot q'(x_2)$. This new Ltw would produce "more to the left", but still be equivalent to the first M.

In order to eliminate these differences in output strategies, we want transducers to produce the output as up in the input tree as possible, and then as to the left as possible. We formalize these notions in the definition of *earliest* Ltws.

To simplify notations, we note $lcp(q)$ (or $lcs(q)$) for $lcp(L_q)$ (or $lcs(L_q)$). By extension, for $u \in \Delta^*$, we note $lcp(qu)$ (or $lcs(qu)$) for $lcp(L_q.u)$ (or $lcs(L_q.u)$).

Definition 7. *A* Ltw M *is* earliest *if it is trimmed, and:*

- *For every state q, $lcp(q) = lcs(q) = \varepsilon$*
- *For each rule $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n \in rul$, for every i from 1 to n, $lcp(q_i u_i) = \varepsilon$*

This definition is a generalization of the one found in [7] from Stws to all Ltws. The first item ensures an earliest Ltw outputs as soon as possible, the second that it produces as to the left as possible. Note that this means that $u_0 q_1(x_{\sigma(1)}) ... q_i(x_{\sigma(i)}) u_i$ produces as much of $[\![M]\!]_q(f(s_1 ... s_n))$ by just knowing $s_{\sigma(1)}, ..., s_{\sigma(i)}$, i.e. the *lcp* of all $[\![M]\!]_q(f(s_1 ... s_n))$ for some fixed $s_{\sigma(1)}, ..., s_{\sigma(i)}$.

Lemma 8. *For M an earliest* Ltw*, $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n \in rul$, for i such that $i \leq n$, $t_{\sigma(1)}, ..., t_{\sigma(i)}$ respectively in $dom([\![M]\!]_{q_1}), ..., dom([\![M]\!]_{q_i})$, then $u_0 [\![M]\!]_{q_1}(t_{\sigma(1)}) ... [\![M]\!]_{q_i}(t_{\sigma(i)}) u_i$ is the lcp of the set:*

$$\left\{ [\![M]\!]_q(f(s_1, ...s_n)) | s_{\sigma(1)} = t_{\sigma(1)}, ..., s_{\sigma(i)} = t_{\sigma(i)} \right\}$$

.

In intuition, this comes from the fact that in an earliest, on the right of $u_0 [\![M]\!]_{q_1}(t_{\sigma(1)}) ... [\![M]\!]_{q_i}(t_{\sigma(i)}) u_i$, one cannot guess the first letter of $[\![M]\!]_{q_{i+1}}(t_{\sigma(i+1)}) ... [\![M]\!]_{q_n}(t_{\sigma(n)}) u_n$.

Some important properties extend from [7] to earliest Ltws, most notably the fact that all Ltws can be made earliest.

Lemma 9. *For M a* Ltw*, one can compute an equivalent earliest* Ltw *in exponential time.*

This result is a direct generalization of the construction in Section 3 of [7]. We build the equivalent earliest Ltw M' with two kinds of steps:

- If $lcp(qu) = v$, where v is a prefix of u, we can slide v through state q by creating a new state $\left[v^{-1} q v\right]$ such that for all t, $[\![M']\!]_{[v^{-1} q v]}(t) = v^{-1} [\![M]\!]_q(t) v$. Every occurrence of $q(x_i)v$ in a rule of M is replaced by $v \left[v^{-1} q v\right](x_i)$.

– If $lcp(q) = v$, we can produce v outside of q by creating a new state $\left[v^{-1}q\right]$ such that for all t, $[\![M']\!]_{[v^{-1}q]}(t) = v^{-1}[\![M]\!]_q(t)$. Every occurrence of $q(x_i)$ in a rule of M is replaced by $v\left[v^{-1}q\right](x_i)$.
Symmetrically, if $lcs(q) = v$, we create a state $\left[qv^{-1}\right]$, and every occurrence of $q(x_i)$ in a rule of M is replaced by $\left[qv^{-1}\right](x_i)v$.

Note that the exponential bound is, in fact, an exact bound, as some LTWs gain an exponential number of states through this process.

In [7], earliest STWs are actually enough to make a normal form using a Myhill-Nerode theorem: by minimizing earliest STWs (merging states with the same $[\![M]\!]_q$), we end up with a normal form with a minimal number of states. However, in the wider case of LTWs, there are still ways for two states to be equivalent and yet not syntactically equal. This impedes the process of minimization. As we will see in the next part, it remains to study how the images of subtrees can be reordered in earliest LTWs while preserving equivalence.

4 Reordering in Earliest Transducers

Syntactically different earliest LTWs may still be equivalent. Indeed, unlike sequential tree transducers [7], which impose the output to follow the order of the input, LTWs permit to flip the order.

The main point of this paper is the observation that it is sufficient to normalize the flips in the output production of earliest LTWs, in order to find a unique normal form for equivalent LTWs. To this end, we will prove that order differences are only possible in very specific cases. We start illustrating such flips in some examples, and then discuss the necessary and sufficient condition that dictates when a flip is possible.

Example 10. We reconsider Example 6 . This earliest transducer "counts" the number of nodes in the input tree has only one state q'. It has the axiom $ax' = 0 \cdot q'$ and the following rules:

$$q'(f(x_1, x_2)) \to 0 \cdot 0 \cdot q'(x_1) \cdot q'(x_2), \qquad q'(a) \to \varepsilon, \qquad q'(b) \to \varepsilon.$$

We can now flip the order of the terms $q'(x_2)$ and $q'(x_1)$ in the first rule, and replace it by:

$$q'(f(x_1, x_2)) \to 0 \cdot 0 \cdot q'(x_2) \cdot q'(x_1).$$

This does not change $[\![M']\!]$, since just the order is changed in which the nodes of the first and second subtree of the input are counted.

Of course, it is not always possible to flip two occurrences of terms $q_1(x_{\sigma(1)})$ and $q_2(x_{\sigma(2)})$ in LTW rules.

Example 11. Consider an earliest transducer that outputs the frontier of the input tree while replacing a by 0 and b by 1. This transducer has a single state q, the axiom $ax = q$, and the following rules:

$$q(f(x_1, x_2)) \to q(x_1) \cdot q(x_2), \qquad q(a) \to 0, \qquad q(b) \to 1.$$

Clearly, replacing the first rule by a flipped variant $q(f(x_1, x_2)) \to q(x_2) \cdot q(x_1)$ would not preserve transducer equivalence since $f(a, b)$ would be transformed to 10 instead of 01. More generally, no LTW with rule $q(f(x_1, x_2)) \to u_0 \cdot q_1(x_2) \cdot u_1 \cdot q_2(x_1) \cdot u_2$ produces the correct output.

Our goal is to understand the conditions when variable flips are possible.

Definition 12. *For M, M' two* LTWs*, $q \in Q$, $q' \in Q'$,*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n \in rul$$
$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n \in rul'$$

are said to be twin rules *if q and q' are equivalent.*

4.1 Reordering Erasing States

We start the study of possible reordering with the obvious case of states that only produce ε: they can take every position in every rule without changing the semantics of the states. The first step towards normalization would then be to fix the positions of erasing states in the rules, to prevent differences in equivalent earliest LTWs: we put all erasing states at the end of any rule they appear in, in ascending subtree order.

Definition 13. *For M a* LTW*, a state q is* erasing *if for all $t \in dom([\![M]\!]_q)$, $[\![M]\!]_q(t) = \varepsilon$*

We show that if two states are equivalent, they call erasing states on the same subtrees. We start by this length consideration:

Lemma 14. *For two twin rules of earliest* LTWs

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$
$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n$$

For i, j such that $\sigma(i) = \sigma'(j)$, and $t_{\sigma(i)} \in dom([\![M]\!]_{q_i})$ then $|[\![M]\!]_{q_i}(t_{\sigma(i)})| = |[\![M']\!]_{q'_i}(t_{\sigma(i)})|$

Proof. The equivalence of q and q' gives for all $t_1, ..., t_n$:

$$u_0 [\![M]\!]_{q_1}(t_{\sigma(1)}) ... [\![M]\!]_{q_n}(t_{\sigma(n)}) u_n = u_0 [\![M']\!]_{q'_1}(t_{\sigma'(1)}) ... [\![M']\!]_{q'_n}(t_{\sigma'(n)}) u'_n$$

By fixing every t_k except $t_{\sigma(i)}$ we get that for some u, v, u', v', $u[\![M]\!]_{q_i}(t_{\sigma(i)})v = u'[\![M']\!]_{q'_j}(t_{\sigma(i)})v'$. If $|[\![M]\!]_{q_i}(t_{\sigma(i)})| > |[\![M']\!]_{q'_j}(t_{\sigma(i)})|$ then $|u| < |u'|$, or $|v| < |v'|$. If $|u| < |u'|$, then $u' = uw$. For all $t_{\sigma(i)}$, $[\![M]\!]_{q_i}(t_{\sigma(i)}) \neq \varepsilon$ (it is longer than $[\![M']\!]_{q'_j}(t_{\sigma(i)})$), and its first letter is always the first letter of w. This means $lcp(q_i) \neq \varepsilon$, which is impossible in an earliest LTW. $|v| < |v'|$ leads to $lcs(q_i) \neq \varepsilon$, another contradiction. By symmetry, $|[\![M']\!]_{q'_j}(t_{\sigma(i)})| > |[\![M]\!]_{q_i}(t_{\sigma(i)})|$ also leads to contradiction. Therefore, both are of same size.

Lemma 15. *For two twin rules of earliest* LTWs

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$
$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n$$

For i, j such that $\sigma(i) = \sigma'(j)$, If q_i is erasing, then q'_j is erasing.

To normalize the order of erasing states in twin rules, we note that since an erasing state produces no output letter, its position in a rule is not important to the semantics or the earliest property. We can thus push them to the right.

Lemma 16. *For M an earliest* Ltw*, $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n$ a rule in M, and q_i an erasing state. Then replacing this rule by*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots u_{i-1} u_i \dots q_n(x_{\sigma(n)})u_n q_i(x_{\sigma(i)})$$

does not change $[\![M]\!]_q$, and M remains earliest.

Note that the earliest property also imposes that if q_i is erasing, $u_i = \varepsilon$.

Given this lemma, we can define a first normalization step where all erasing states appear at the end of the rules in ascending subtree order.

Definition 17. *An earliest* Ltw *M is* erase-ordered *if for every rule $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n \in rul$, if q_i is erasing, then for all $j > i$, q_j is erasing, and $\sigma(i) < \sigma(j)$.*

Lemma 18. *For M an earliest* Ltw*, one can make M erase-ordered in polynomial time without changing the semantic of its states.*

We can detect if a state q is erasing by checking that no accessible rule produces a letter. From there, Lemma 16 ensures that making a Ltw erase-ordered is just a matter of pushing all erasing states at the end of the rules and them sorting them in ascending subtree order.

4.2 Reordering Producing States

As we saw in Example 11, some flips between states are not possible. We will now study what makes reordering non-erasing states possible. As we will see, only few differences are possible between twin rules in erase-ordered earliest Ltws. Two states transforming the same subtree are equivalent, and the only order differences are caused by flipping states whose productions commute in Δ^*.

To prove this, we begin by establishing a few preliminary results. We first show that to the left of σ and σ''s first difference, both rules are identical.

Lemma 19. *For two twin rules of erase-ordered earliest* Ltw*s M, M'*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n$$
$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)})u'_n$$

For i such that if $k \leq i$ then $\sigma(k) = \sigma'(k)$, $[\![M]\!]_{q_i} = [\![M']\!]_{q'_i}$, and $u_i = u'_{i'}$

Proof. This results from Lemma 8: if σ and σ' coincide before i, then for all $t_{\sigma(1)}, ..., t_{\sigma(i)}$, $u_0[\![M]\!]_{q_1}(t_{\sigma(1)})...u_i$ and $u'_0[\![M']\!]_{q_1}(t_{\sigma'(1)})...u'_i$ are both equal to the *lcp* of $\{[\![M]\!]_q(f(s_1, ..., s_n)) | s_{\sigma(1)} = t_{\sigma(1)}, ..., s_{\sigma(n)} = t_{\sigma(n)}\}$. This means that:

$$u_0[\![M]\!]_{q_1}(t_{\sigma(1)})...[\![M]\!]_{q_i}(t_{\sigma(i)})u_i = u_0[\![M']\!]_{q'_1}(t_{\sigma'(1)})...[\![M']\!]_{q'_i}(t_{\sigma'(i)})u'_i$$

Since this is also true for $i-1$, we can remove everything but the last part for each side of this equation, to obtain that for all $t_{\sigma(i)}$, $[\![M]\!]_{q_i}(t_{\sigma(i)})u_i = [\![M']\!]_{q'_i}(t_{\sigma(i)})u'_i$. Lemma 14 gives us $|[\![M]\!]_{q_i}(t_{\sigma(i)})| = |[\![M']\!]_{q'_i}(t_{\sigma'(i)})|$, and $u_i = u'_i$. This means that q_i and q'_i are equivalent, and $u_i = u'_i$.

It still remains to show what happens when σ and σ' stop coinciding. We study the leftmost order difference between two twin rules in erasing-ordered earliest LTWs, that is to say the smallest i such that $\sigma(i) \neq \sigma'(i)$. Note that Lemma 15 ensures that such a difference occurs before the end of the rule where the erasing states are sorted.

Lemma 20. *For two twin rules of erase-ordered earliest* LTW*s* M, M'

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$
$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n$$

For i such that $\sigma(i) \neq \sigma'(i)$ and for any $k < i$, $\sigma(k) = \sigma'(k)$, for j such that $\sigma'(i) = \sigma(j)$, we have:

(A) For all k from i to $j-1$, $u_k = \varepsilon$ and there exists $t^{\varepsilon}_{\sigma(k)}$ such that $[\![M]\!]_{q_k}(t^{\varepsilon}_{\sigma(k)}) = \varepsilon$
(B) For all k from i to j, for k' such that $\sigma(k) = \sigma'(k')$, q_k is equivalent to $q'_{k'}$
(C) All $q_i, ..., q_j$ are periodic of same period.

As a proof intuition, we first prove point (A), then use it to show point (B), then from (A) and (B) we finally show point (C).

For point (A), we use the equivalence of q and q'. For all $t_1, ..., t_n$,

$$u_0[\![M]\!]_{q_1}(t_{\sigma(1)})...[\![M]\!]_{q_n}(t_{\sigma(n)})u_n = u_0[\![M']\!]_{q'_1}(t_{\sigma'(1)})...[\![M']\!]_{q'_n}(t_{\sigma'(n)})u'_n$$

Lemma 19 gives us that everything up to u_{i-1} and u'_{i-1} coincide. We then get

$$[\![M]\!]_{q_i}(t_{\sigma(i)})...[\![M]\!]_{q_n}(t_{\sigma(n)})u_n = [\![M']\!]_{q'_i}(t_{\sigma'(i)})...[\![M']\!]_{q'_n}(t_{\sigma'(n)})u'_n$$

Since q'_i is not erasing, we can fix $t_{\sigma'(i)}$ such that $[\![M']\!]_{q'_i}(t_{\sigma'(i)}) \neq \varepsilon$. We call its first letter a. All non-ε productions of q_i must begin by a. This is only possible in an earliest if there exists $t^{\varepsilon}_{\sigma(i)}$ such that $[\![M]\!]_{q_i}(t^{\varepsilon}_{\sigma(i)}) = \varepsilon$. We now fix $t_{\sigma(i)} = t^{\varepsilon}_{\sigma(i)}$. If $u_i \neq \varepsilon$, its first letter is a. This is impossible in an earliest since it would mean $lcp(q_i u_i) \neq \varepsilon$. Hence $u_i = \varepsilon$ We can make the same reasoning for q_{i+1} and u_{i+1}, and so on all the way to q_{j-1} and u_{j-1}.

For point (B), we use point (A) to eliminate everything in front of q_k and $q'_{k'}$ by picking all $t^{\varepsilon}_{\sigma(l)}$ up to $k-1$ and all $t^{\varepsilon}_{\sigma'(l')}$ up to $k'-1$.

$$[\![M]\!]_{q_k}(t_{\sigma(k)})...[\![M]\!]_{q_n}(t_{\sigma(n)})u_n = [\![M']\!]_{q'_{k'}}(t_{\sigma'(k')})...[\![M']\!]_{q'_n}(t_{\sigma'(n)})u'_n$$

From Lemma 14, we know that $|[\![M]\!]_{q_k}(t_{\sigma(k)})| = |[\![M']\!]_{q'_{k'}}(t_{\sigma(k)})|$. We conclude that q_k and $q'_{k'}$ are equivalent.

For point (C), we take k' such that $\sigma(k) = \sigma'(k')$. We use (A) to erase everything but q_k, q_j, q'_i and $q'_{k'}$ by picking every $t^{\varepsilon}_{\sigma(l)}$ and $t^{\varepsilon}_{\sigma'(l')}$ except theirs.

$$[\![M]\!]_{q_k}(t_{\sigma(k)})[\![M]\!]_{q_j}(t_{\sigma(j)})...u_n = [\![M']\!]_{q'_i}(t_{\sigma'(i)})[\![M']\!]_{q'_{k'}}(t_{\sigma'(k')})...u'_n$$

Point (B) gives q_k is equivalent to $q'_{k'}$ and q_j is equivalent to q'_i. We get that $[\![M]\!]_{q_k}(t_{\sigma(k)})[\![M]\!]_{q_j}(t_{\sigma(j)}) = [\![M]\!]_{q_j}(t_{\sigma(j)})[\![M]\!]_{q_k}(t_{\sigma(k)})$. This means that the productions of q_k and q_j commute, which in Δ^* is equivalent to say they are words of same period. Therefore, q_j and q_k are periodic of same period.

This result allows us to resolve the first order different between two twin rules by flipping q_j with neighbouring periodic states of same period. We can iterate this method to solve all order differences.

Theorem 21. *For two twin rules of erase-ordered earliest* LTWs*,*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$
$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n$$

One can replace the rule of q to another rule of same subtree order as the rule of q' only by flipping neighbour states q_k and q_{k+1} of same period where $u_k = \varepsilon$.

We can use Lemma 20 to solve the leftmost difference: for i first index such that $\sigma(i) \neq \sigma'(i)$, and j such that $\sigma(i) = \sigma'(j)$, we have $u_i = ... = u_{j-1} = \varepsilon$ and $q_i, ..., q_j$ commute with each other. This means we can replace the first rule by:

$$q, f \to u_0 ... q_j(x_{\sigma(j)}) q_i(x_{\sigma(i)}) ... q_{j-1}(x_{\sigma(j-1)}) u_j ... u_n$$

where $q_j(x_{\sigma(j)})$ is to the left of $q_i(x_{\sigma(i)}) ... q_{j-1}(x_{\sigma(j-1)})$ without changing $[\![M]\!]_q$.

This solves the leftmost order difference: we can iterate this method until both rules have the same order.

Finally, we call Lemma 19 on the rules reordered by Theorem 21 to show that two twin rules use equivalent states and the same constant words:

Theorem 22. *For two twin rules of erase-ordered earliest* LTWs*,*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$
$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n$$

$u_0 = u'_0, ..., u_n = u'_n$*, and for k, k' such that $\sigma(k) = \sigma'(k')$, $[\![M]\!]_{q_k} = [\![M']\!]_{q'_{k'}}$.*

5 Myhill-Nerode Theorem and Normal Form

In Section 3, we showed that LTWs can be made earliest. In Section 4, we first showed that all earliest LTWs can be made erase-ordered, then we made explicit what reorderings are possible between two rules of two equivalent states. In this section, we use these results to fix a reordering strategy. This will give us a new normal form, *ordered earliest* LTWs. We will show that each LTW in equivalent to a unique minimal ordered earliest LTW, whose size is at worst exponential.

We first use Theorem 21 to define a new normal form: ordered earliest LTWs.

Definition 23. *A* LTW *M is said to be* ordered earliest *if it is earliest, and for each rule $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$:*

- *If q_i is erasing, then for any $j > i$, q_j is erasing.*
- *If $u_i = \varepsilon$, and q_i and q_{i+1} are periodic of same period, $\sigma(i) < \sigma(i+1)$.*

Note that this definition notably implies that any ordered earliest is erase-ordered earliest. On top of that, we impose that if two adjacent states are periodic of same period, and thus could be flipped, they are sorted by ascending subtree.

Lemma 24. *For M an earliest* LTW*, one can make M ordered in polynomial time without changing the semantic of its states.*

We saw in Lemma 18 that one can push and sort erasing states. For this result, sorting periodic states is not more complicated. However, one must test first whether two states are periodic of same period. This can be done in polynomial time. One can prove that the productions of a LTW state q form an algebraic language (described by a context-free grammar). Then, the problem of deciding if two algebraic languages are periodic of same period is known to be polynomial.

Our goal is now to show the existence of a unique minimal normal LTW equivalent to any M. To this end, we first show that two equivalent LTW*s* will use the same states: any $q \in Q$ has an equivalent $q' \in Q'$.

Lemma 25. *For two equivalent earliest* LTW*s* M *and* M'*, for* q *state of* M*, there exist an equivalent state* q' *in* M'*.*

Proof. We start by the axioms: if $ax = u_0 q_0 v_0$ and $ax' = u'_0 q'_0 v'_0$, since M and M' are earliest, $u_0 = lcp([\![M]\!]) = lcp([\![M']\!]) = u'_0$. Then, $v_0 = lcs(q_0 v_0) = lcs(q'_0 v'_0) = v'_0$. We then get that q_0 and q'_0 are equivalent.

We can then call Theorem 22 to twin rules of equivalent states q, q' to get new equivalent pairs q_k, $q'_{k'}$ for $\sigma(k) = \sigma'(k')$. Since M is trimmed, this recursive calls will eventually reach all $q \in Q$ and pair them with an equivalent $q' \in Q'$.

Since all equivalent earliest LTWs use the same states, they have the *minimal* amount of states when they don't have two redundant states q, q' such that $[\![M]\!]_q = [\![M]\!]_{q'}$. We show this characterises a *unique minimal normal form.*

Theorem 26. *For* M *a* LTW*, there exists a unique minimal ordered earliest* LTW M' *equivalent to* M *(up to state renaming).*

The existence of such a minimal ordered earliest LTW derives directly from Lemma 24. All we need to make an ordered earliest M' minimal is to merge its equivalent states together, which is always possible without changing $[\![M']\!]$.

The uniqueness derives from several properties we showed in this paper . Imagine M and M' two equivalent minimal ordered earliest LTWs. The fact that they have equivalent states come from Lemma 25. Since both are minimal, neither have redundant state: each q of M is equivalent to exactly one q' of M' and vice-versa. From Theorem 22, we know that two equivalent states call equivalent states in their rules, with only the possibility of reordering periodic states. Since M and M' are ordered, twin rules also have same order.

6 Conclusion and Future Work

This paper 's goal was to solve the equivalence problem on linear tree-to-word transducers, by establishing a normal form and a Myhill-Nerode theorem on this class. To do so we naturally extended the notion of earliest transducers that already existed in sequential tree transducers [7]. However it appeared that this was no longer enough to define a normal form: we studied all possible reorderings that could happen in an earliest LTW. We then used this knowledge to define a

new normal form, that has both an output strategy (earliest) and an ordering strategy (ordered earliest), computable from any LTW in EXPTIME.

There are several ways to follow up on this result: one would be adapting the learning algorithm presented in [8], accounting for the fact that we now also have to learn the order in which the images appear. It could also be relevant to note that in [7], another algorithm decides equivalence in polynomial time, which is more efficient than computing the normal form. Such an algorithm would be an improvement over the actual randomized polynomial algorithm by [12]. As far as Myhill-Nerode theorems go, the next step would be to consider all tree-to-word transducers. This problem is known to be difficult. Recently, [12] gave a randomized polynomial algorithm to decide equivalence, but did not provide a Myhill-Nerode characterization.

References

1. Rajeev Alur and Loris D'Antoni. Streaming tree transducers. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, Roger Wattenhofer, Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP (2)*, volume 7392 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2012.
2. Christian Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131–143, 2003.
3. J. Engelfriet and S. Maneth. The equivalence problem for deterministic MSO tree transducers is decidable. In R. Ramanujam and S. Sen, editors, *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science – FSTTCS'2005*, volume 3821 of *LNCS*, pages 495–504. Springer-Verlag, 2005.
4. Joost Engelfriet and Sebastian Maneth. Macro tree translations of linear size increase are MSO definable. *SIAM Journal on Computing*, 4(32):950–1006, 2003.
5. Joost Engelfriet, Sebastian Maneth, and Helmut Seidl. Deciding equivalence of top-down XML transformations in polynomial time. *Journal of Computer and System Science*, 75(5):271–286, 2009.
6. Joost Engelfriet and Heiko Vogler. Macro tree transducer. *Journal of Computer and System Science*, 31:71–146, 1985.
7. Grégoire Laurence, Aurélien Lemay, Joachim Niehren, Slawek Staworko, and Marc Tommasi. Normalization of sequential Top-Down Tree-to-Word transducers. In Adrian H. Dediu, Shunsuke Inenaga, Carlos M. Vide, Adrian H. Dediu, Shunsuke Inenaga, and Carlos M. Vide, editors, *LATA*, volume 6638 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2011.
8. Grégoire Laurence, Aurélien Lemay, Joachim Niehren, Slawek Staworko, and Marc Tommasi. Learning sequential tree-to-word transducers. In Adrian Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, volume 8370 of *Lecture Notes in Computer Science*, pages 490–502. Springer, 2014.
9. Aurélien Lemay, Sebastian Maneth, and Joachim Niehren. A Learning Algorithm for Top-Down XML Transformations. In Acm, editor, *29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 285–296, Indianapolis, United States, 2010. ACM Press.

10. Sebastian Maneth, Thomas Perst, and Helmut Seidl. Exact XML type checking in polynomial time. In *International Conference on Database Technology*, volume 4353 of *Lecture Notes in Computer Science*, pages 254–268. Springer Verlag, 2007.
11. J. Oncina, P. Garcia, and E. Vidal. Learning subsequential transducers for pattern recognition and interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:448–458, 1993.
12. Helmut Seidl, Sebastian Maneth, and Gregor Kemper. Equivalence of deterministic top-down tree-to-string transducers is decidable. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 943–962. IEEE Computer Society, 2015.