Sequential Tree-to-Word Transducers: Normalization, Minimization, and Learning

Grégoire Laurence[a,c], Aurélien Lemay[a,c], Joachim Niehren[a,d], Sławek Staworko[a,c], Marc Tommasi[b,c]

[a] *Links project, INRIA & CRIStAL (CNRS UMR9183)*
[b] *Magnet project, INRIA & CRIStAL (CNRS UMR9183)*
[c] *University of Lille*
[d] *INRIA, Lille*

Abstract

We introduce a class of deterministic sequential top-down tree-to-word transducers (STWs) and investigate a number of its fundamental properties and related problems. First, we investigate the problem of *normalization* of STWs: we identify a subclass of *earliest* STWs (eSTWs) that is as expressive as STWs and present an effective procedure for converting an arbitrary STW into an equivalent eSTW. We then present a *Myhill-Nerode* characterization of the class of the transformations definable with STWs which also shows that every transformation defined with an STW has a unique canonical representative eSTW. This canonical eSTW is the minimal eSTW defining the same transformation, and consequently, we present a polynomial minimization procedure for eSTWs, thus giving an effective procedure for constructing the canonical representative of any transformation definable with STWs. Finally, we use the Myhill-Nerode characterization to devise an algorithm for inference (learning) of eSTWs from examples of transformation given by the user.

Keywords: Language Theory, Transducers, Grammatical Inference

1. Introduction

The classical problems relating to transducers are equivalence, minimization, learning, type checking, and functionality [2, 22, 23, 12]. 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 [15]. We thus follow this tradition to study classes of deterministic transducers. The problems of equivalence, minimization, and learning are often solved using a unique normal representation of transformations definable with a transducer from a given class [16, 13, 11, 20]. Such normal forms are typically obtained with the help of a Myhill-Nerode characterization, which is of independent interest and has a number of important applications.

In this paper, we investigate the class of deterministic sequential top-down tree-to-word transducers (STWs) and study the problems of normalization, minimization, and inference for this class of transducers. 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 lays in the fact that tree-to-word transformations are better suited to model general XML transformations as opposed to tree-to-tree transducers [11, 20, 23]. 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 [12, 27, 31].

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 tree. Nevertheless STWs are very powerful and 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 [11, 20].

Unique normal forms of transducers are typically obtained in two steps: output normalization followed by 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 [13].

We propose a natural normal form for STWs based on being earliest and define the corresponding class of *earliest* STWs (eSTWs) using easy to verify structural requirements. The first contribution of the present paper is 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 comes from the fact that the output language of an STW may be an arbitrary context-free language.

Then, we investigate the problem of minimizing eSTWs and show that it 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, which further validates the proposed normal form for STWs. Next, we present a Myhill-Nerode characterization of transformations definable with STWs. It proves that the result of normalization of an STW followed by subsequent minimization of the obtained eSTW yields a unique *canonical* eSTW representative. The Myhill-Nerode characterization is in fact a semantic characterization of tree-to-word transformation that can be captured with STWs and provides a direct way of constructing the canonical eSTW using the notion of residual of a transformation and a Myhill-Nerode equivalence relation on paths of the input trees.

Finally, we use the Myhill-Nerode characterization to devise a learning algorithm for eSTW from examples of a tree-to-word transformation given by the user. The algorithm is essentially an adaptation of the construction of canonical transducer to the setting where only a finite fragment of the transformation is know, in the form of a finite set of *examples* consisting of the input tree and the corresponding output word. One of the consequences of the Myhill-Nerode characterization is that for any transformation definable with an eSTW, there exists a *characteristic* set of examples for which the learning algorithm constructs the correct transducer, and furthermore, we show that there exists a characteristic set of examples whose number is polynomial in the size of the goal transducer. It must be pointed out, however, that inference of eSTWs from examples is not a trivial matter: given a set of examples it is NP-complete to decide if there even exists an eSTW that is *consistent* with the given examples. Consequently, we propose a novel learning model where the learning algorithm may *abstain* from constructing a consistent transducer, thus indicating to the user that more examples are necessary. However, once the user provides sufficiently informative examples (i.e., a characteristic set of examples), the algorithm is no longer allowed to abstain, which prevents the learning setting for admitting trivial solutions of learning algorithms that always abstain.

Related Work. The problems of normalization, minimization, Myhill-Nerode characterization, and inference are fundamental problems that have been studied for a large number of classes of transducers. Rational transducers, which are essentially automata whose transitions additionally specify output words, are the basic class of finite machines used to model word-to-word transformations. However, in their generality they are non-deterministic, which precludes the existence of a unique normal form and unique minimal transducers for the same reason why a non-deterministic word automaton does not have in general a unique minimal equivalent automaton [17]. Subsequential transducers are rational transducers that are deterministic w.r.t. the input word and this restriction enables a normal form, called the *onward form* (cf. [7] for a survey). It basically requires the output to be produced as soon as possible in the process of reading the input. In fact, one can consider this development to be the original *earliest* normal form at which normal forms for a number of classes of transducers take roots, for instance the normal form for the class of functional rational transformations [28]. The normal form for subsequential transducers is also a clear inspiration to normal forms for tree-to-tree transducers, such as the class of deterministic top-down transducers for which a normal form has been presented in [11], which has further led to a Myhill-Nerode characterization [20]. Also the normal form for the class of bottom-up tree-to-tree transducers [13], and the corresponding Myhill-Nerode characterization, has been originally inspired by the earliestness approach, however the normal form needs to account for the specific bottom-up operation of this class of transducers, and in the end, it is not an exact instance of the earliestness approach.

Naturally, the normal form for STWs that we propose in this paper is inspired by the normal form for subsequent transducers, however, it should be pointed out that it is not an adaptation of the normal forms for the classes of tree-to-tree transducers discussed above. This is because, in general, tree-to-word transducers generalize tree-to-tree transducers because the former can produce a word that is a serialization of a tree. We point out, however, that the restrictions imposed by the class of STW i.e., each node of the input tree is visited exactly once in a fixed predefined order, make the class incomparable to the classes of tree-to-tree transducers discussed above.

Existence of normal forms for more complex classes is generally an open question, as for instance for the class of Macro Tree Transducers (MTTs) [10], which are essentially top-down tree-transducers with an output that can use accumulators (macros). The use of accumulators is a generalization of output concatenation and as such STWs can be seen as a very specific restricted subclass of MTTs. This suggest that a normal form for MTTs, if it exists, will be considerably more complex and constructing it will very likely pose a much higher challenge. Our results also offer a step towards a better understanding of the challenges in formulating a normal form for the class of deterministic nested-word-to-word transducers (dN2Ws) [12] 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 [31]. We point out, however, that there exist arguments suggesting that arbitrary dN2Ws are unlikely to have natural normal forms [1].

The approach we employ for learning eSTWs subscribes to general area of grammatical inference. Originally proposed by Gold [14] for languages of words, and later on adapted to a number of other concepts including inference of regular languages of words and trees [24, 26] (see also [9] for survey of the area), learning of DTDs and XML Schema [4, 3], and XML queries [6, 30]. The existence of a unique normal form, typically obtained with the help of a Myhill-Nerode characterization, has been the basis of learning algorithms for a number of classes of transducers, including subsequential transducers [25], functional rational transducers [5], and top-down tree-to-tree transducers [20]. While a normal unique form and a Myhill-Nerode characterization for bottom-up tree-to-tree transducers exists [13], the existence of a suitable learning algorithm remains an open question. Inference of more general Macro Tree Transducers is also an open question.

Organization. Section 2 contains the basic notions and notations we use throughout the paper. In Section 3 we present the class of deterministic sequential tree-to-word transducers (STWs) and define their normal form, the earliest deterministic sequential tree-to-word transducers (eSTWs). In Section 4 we investigate the normalization problem i.e., converting an arbitrary STW to an equivalent eSTW and present an effective normalization procedure. In Section 5 we present a Myhill-Nerode characterization of transformation definable with STWs, investigate the problem of minimizing eSTWs (and STWs), and show that the unique canonical representative of a transformation defined by an arbitrary STW is the minimal equivalent eSTW, which can be effectively constructed by normalizing the given STW and then minimizing the intermediate eSTW. In Section 6 we investigate the problem of learning of eSTWs from a finite set of examples. Finally, in Section 7 we summarize our work and outline future directions. A number of results in this paper is of combinatorial nature and while their validity is relatively easy to see, their proofs require detailed and technical arguments. Consequently, for the sake of the presentation, we briefly outline a number of proofs and present detailed versions in the appendix.

2. Preliminaries

We begin by recalling a number of standard notions. More detailed presentation can be found, for instance, in [8].

Words. 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 $w \cdot v^{-1} = u$.

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. We assume that every symbol has a unique arity, i.e. $\Sigma^{(i)} \cap \Sigma^{(j)} = \emptyset$ for $i \neq j$. We sometimes write $f^{(k)}$ to indicate explicitly that $f \in \Sigma^{(k)}$. A *tree* is a ranked ordered term over Σ and we use $t, t_0, t_1, \ldots$ to range over trees. For instance, $t_0 = f(a, g(b))$ is a tree over $\Sigma = \{f^{(2)}, g^{(1)}, a^{(0)}, b^{(0)}\}$. By T_Σ we denote the set of all trees over Σ. A tree language is a subset of T_Σ. We use $T, T_0, \ldots$ to range over tree languages.

Paths. A (*labeled*) *path* is a word over $\bigcup_{k>0} \Sigma^{(k)} \times \{1, \ldots, k\}$, which identifies a node in a tree by the positions and 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 *paths*(t) we denote the set of paths of a tree t. For instance, for $t_0 = f(a, g(b))$ we get *paths*$(t_0) = \{\varepsilon, (f,1), (f,2), (f,2) \cdot (g,1)\}$. Similarly, for a set of trees T, *paths*$(T) = \bigcup_{t \in T}$ *paths*(t).

Contexts. A *tree context* over Σ is a tree over $\Sigma \cup \{x^{(0)}\}$ having exactly one occurrence of a distinguished placeholder x, a symbol of arity 0 not present in Σ. We use $c, c_0, c_1, \ldots$ to range over tree contexts (c is also used as an example of symbol in Δ but this use never leads to confusion) and $C, C_0, C_1, \ldots$ to range over sets of contexts. By C_Σ we denote the set of all contexts over Σ. By $c[t]$ we denote the result of replacing x with t. For instance, if we take the context $c_0 = f(g(a), x)$, then $c_0[t_0] = f(g(a), f(a, g(b)))$.

Canonical orders. 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. A precise example of such orders is given in the appendix.

Transformations. Given a ranked tree alphabet Σ and a finite set of symbols Δ, a *tree-to-word transformation* (or simply *transformation*) is a possibly partial function τ that maps trees over Σ to words over Δ. By $dom(\tau)$ we denote the *domain* of τ i.e., set of all trees for which τ is defined and $ran(\tau)$ is the *range* of τ i.e., the set of all words returned by τ. In the sequel, we work with transformations having a nonempty domain only and use $\tau, \tau_0, \tau_1, \ldots$ to range over transformations.

Deterministic top-down tree automata. The transducer model that we propose and study in this paper is based on, and can be viewed as a natural extension of, deterministic top-down tree automata which we recall next. Formally, a *deterministic top-down tree automaton* is a tuple $A = (\Sigma, Q, q_0, \delta)$, where Σ is a ranked alphabet of *input trees*, Q is a finite set of states, $q_0 \in Q$ is the *initial state* and δ is a partial *transition function* from $Q \times \Sigma$ to Q^* such that if $\delta(q, f^{(k)})$ is defined, then it belongs to Q^k. Rather than using the notion of a run to define the semantics of DTAs, we employ a recursive definition that we adapt later to define the semantics of STWs. Essentially, for every state $q \in Q$ we define the set of trees $[\![A]\!]_q$ recognized by q. We bind those sets with the following set of mutually recursive assertions (with $q \in Q$):

$$f(t_1, \ldots, t_k) \in [\![A]\!]_q \iff \delta(q, f) = q_1 \cdot \ldots \cdot q_k \text{ and } \forall i \in \{1, \ldots, k\}\ t_i \in [\![A]\!]_{q_i}.$$

Then, the *language* defined by A is $[\![A]\!] = [\![A]\!]_{q_0}$. The *size* of A, denoted $|A|$, is the sum of the number of states of A and the sizes of the rules of A, where the size of the rule $\delta(q, f) = q_1 \cdot \ldots \cdot q_k$ is $k + 2$.

3. Sequential tree-to-word transducers

The model of transducer we propose is essentially an extension of the deterministic tree automata whose rules additionally indicate the words to be produced in the output (which are concatenated according to the standard left-to-right traversal of the input tree).

Definition 3.1 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*, 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 by STWs the class of deterministic sequential top-down tree-to-word transducers and by $\mathcal{STW}$ 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:

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

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. The *size* of an STW is the sum of the number the states and the lengths of the rules, including the sizes of the words used in rules. More precisely, the size of the rule $\delta(q, f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot q_k \cdot u_k$ is $k + 1 + |u_1| + \ldots + |u_k|$.

Example 3.2 We fix the input alphabet $\Sigma = \{f^{(2)}, g^{(1)}, a^{(0)}\}$ and the output alphabet $\Delta = \{a, b, c\}$. The STW M_1 uses the states $Q = \{q_0, q_1\}$, 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 $[\![M_1]\!](f(g^m(a), g^n(a))) = (abc)^m ac (abc)^n$, where $m, n \geq 0$, and $[\![M_1]\!]$ is undefined on all other input trees.

The STW M_2 has the states $Q = \{q_0, q_1, q_2, q_3\}$, the initial rule q_0, and these transition rules:

$$\begin{array}{llll} \delta(q_0, f) = q_1 \cdot q_3 \cdot ab, & \delta(q_1, g) = a \cdot q_2, & \delta(q_2, g) = ab \cdot q_3, & \delta(q_3, g) = q_3, \\ \delta(q_0, a) = ba, & \delta(q_1, a) = \varepsilon, & \delta(q_2, a) = \varepsilon, & \delta(q_3, a) = \varepsilon. \end{array}$$

Now, $[\![M_2]\!](a) = ba$ and for $n \geq 0$, the result of $[\![M_2]\!](f(g^m(a), g^n(a))$ is ab for $m = 0$, aab for $m = 1$, and $aabab$ for $m \geq 2$; $[\![M_2]\!]$ is undefined for all other input trees. Note that q_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. $\Box$

Recall that a *path* is a word over $\bigcup_{k>0} \Sigma^{(k)} \times \{1, \ldots, k\}$, which identifies a node in an input tree together with the labels of its ancestors: ε is the root node and if a node p is labeled with f, then $p \cdot (f, i)$ is its i-th child. We extend the transition function δ to identify the state reached at a path p: $\delta(q, \varepsilon) = q$ and $\delta(q, p \cdot (f, i)) = q_i$, where $\delta(q, p) = q'$ and $\delta(q', f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot q_k \cdot u_k$.

We say that a state q of an STW M is *productive* if $\mathit{dom}([\![M]\!]_q)$ is non-empty and that q is *accessible* if $\delta(q_0, p) = q$ for some $p \in \mathit{paths}(\mathit{dom}([\![M]\!]))$. In the sequel, w.l.o.g. we consider only *trimmed* STWs, i.e. transducers whose all states are productive and accessible. Naturally, this restriction is merely technical: every STW, that defines a nonempty transformation, can be converted to an equivalent trimmed transducer in time polynomial in the size of the input transducer.

We point out that STWs is a very specific class of tree-to-word transducers and note that in general tree-to-word transducers are more general tree-to-tree transducers because the output word can be used to represent a tree with a simple serialization (cf. Appendix A): indeed, any deterministic top-down tree-to-tree transducer that visits every node of the input tree can be easily represented with an equivalent STW. In fact, STWs can capture arbitrary context-free languages with their output range. On the other hand, while the standard deterministic top-down tree-to-tree transducers [8] can visit a node an arbitrary number of times, they are not capable of capture tree languages representing context-free language. Consequently, STWs and the standard deterministic top-down tree-to-tree transducers are closely related but incomparable. We finish by pointing out that the general model of Macro Tree Transducers can capture STWs with their use of accumulator registers that permits to express concatenation.

Earliest Transducers. Next, we introduce a normal form of the transducers that allows to identify a canonical representative for every STW. The commonly employed method is to require the transducer to produce the output as early as possible. This method has been initially introduced in the context of normalization of the word-to-word transducers [7], and has been successfully applied to classes of tree-to-tree transducers [20, 13]. Adapting this method to STWs is, however, a challenging task, because the output string produced by an STW is obtained in a nontrivial manner, by concatenating the output factors indicated by the rules in the standard preorder left-to-right traversal of the tree. As we illustrate in the following example, a naïve overeager notion of being earliest may result in not every STW having an equivalent early model, and consequently, a more diligent and carefully crafted notion will be necessary.

Example 3.3 Take, for instance, the transformation *turn* that takes a tree over $\Sigma = \{a^{(1)}, b^{(1)}, \perp^{(0)}\}$ and returns the sequence of its labels in the reverse order e.g., $turn(a(b(b(\perp)))) = bba$. It is definable with a simple STW M_{turn}:

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

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 $\perp$, 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. Since the label sequence can be of arbitrary length, a finite memory of any STW is insufficient to store it, 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 M_{turn} in the example above satisfies the condition. We remark, however, 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 3.2) 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. We formalize these conditions as follows.

Definition 3.4 An STW $M = (\Sigma, \Delta, Q, init, \delta)$ is *earliest* (eSTW) iff:

($\mathbf{E_1}$) $lcp(ran(\llbracket M \rrbracket_q)) = \varepsilon$ and $lcs(ran(\llbracket M \rrbracket_q)) = \varepsilon$ for every state q,

($\mathbf{E_2}$) for the initial rule $init = u_0 \cdot q_0 \cdot u_1$ we have $lcp(ran(\llbracket M \rrbracket_{q_0}) \cdot u_1) = \varepsilon$ and for every transition $\delta(q, f) = u_0 \cdot q_1 \cdot \ldots \cdot q_k \cdot u_k$ and every $1 \leq i \leq k$ we have $lcp(ran(\llbracket M \rrbracket_{q_i}) \cdot u_i \cdot \ldots \cdot ran(\llbracket M \rrbracket_{q_k}) \cdot u_k) = \varepsilon$. □

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 $\llbracket M \rrbracket_q$ associated with states have the property that the *lcp* an *lcs* of their output is empty. We note that **($\mathbf{E_1}$)** and **($\mathbf{E_2}$)** can be efficiently checked in an STW because we need only to check that the longest common prefix and suffix of a context-free grammar is the empty word. The empty word is the longest common prefix of a nonempty language if and only if the language contains an empty word or it contains two words that differ on the first position. These conditions can be easily and efficiently checked for context-free languages.

One of the main contribution of the present paper is that, for every STW, there exists a unique equivalent eSTW i.e., its canonical representative, that can be effectively constructed. 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 it thus obtaining the canonical eSTW. We illustrate it on the following example.

Example 3.5 (cont'd. Example 3.2) M_1 is not earliest because **($\mathbf{E_1}$)** 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

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

M_2 is not earliest because **($\mathbf{E_2}$)** is not satisfied by $\delta(q_0, f) = q_1 \cdot q_3 \cdot ab$: every word produced by this rule starts with a. First, we push the word ab through the state q_3, and then we push the symbol a through the state q_1. Pushing through q_3 is easy because it is a deleting state and the rules do not change. Pushing through q_1 requires a recursive push through the states of the rules of q_1 and this process affects the rules of q_2. Finally, we obtain an eSTW with the initial rule q'_0 and the transition rules

$$\begin{array}{llll} \delta'(q'_0, f) = a \cdot q'_1 \cdot b \cdot q'_3, & \delta'(q'_1, g) = a \cdot q'_2, & \delta'(q'_2, g) = ba \cdot q'_3, & \delta'(q'_3, g) = q'_3, \\ \delta'(q'_0, a) = ba, & \delta'(q'_1, a) = \varepsilon, & \delta'(q'_2, a) = \varepsilon, & \delta'(q'_3, a) = \varepsilon. \end{array}$$

□

Finally, we add that the construction of the canonical transducer requires us to formulate a Myhill-Nerode characterization of $\mathcal{STW}$, which we use as a basis for a learning algorithm for the class of eSTWs.

4. Normalization

In this section, we present an effective normalization procedure that converts an STW into an equivalent eSTW. The normalization is the first and the most involved part of the process of constructing a canonical representative of an STW. Once an equivalent eSTW for a given STW is constructed, we minimize it to obtain the canonical representative. The minimization of eSTW is relatively simple and follows the general outline of minimizing deterministic automata thanks to the constraints imposed on earliest STWs.

Normalization of an STW involves changing the placement of the output factors in the transition rules of the transducer in order to satisfy the conditions **(E$_1$)** and **(E$_2$)**, and this process 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, and when investigating the feasibility of the constructions, we consider context-free languages only.

4.1. Reducing languages

Enforcement of **(E$_1$)** corresponds to what we call constructing the *reduced decomposition* of a language (cf. the notions of reduced transformations introduced in Section 5.1). 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 L into its *core* denoted $Core(L)$ and two words $Left(L)$ and $Right(L)$ such that $Core(L)$ is reduced and

$$L = Left(L) \cdot Core(L) \cdot Right(L). \qquad (1)$$

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, $\mathit{Left}(L) = \mathit{lcp}(L)$ and $\mathit{Right}(L) = \mathit{lcs}(L')$, where $L = \mathit{Left}(L) \cdot L'$. Then $\mathit{Core}(L)$ is obtained from (1). As an example, the reduced decomposition of $L_{q_0} = \mathit{ran}([\![M_1]\!]_{q_0}) = (abc)^* ac(abc)^*$ from Example 3.2 is $\mathit{Left}(L_{q_0}) = a$, $\mathit{Right}(L_{q_0}) = c$, and $\mathit{Core}(L_{q_0}) = (bca)^*(cba)^*$.

It is a folklore result that if L is context free, then $\mathit{lcp}(L)$ (and $\mathit{lcs}(L)$) can be of size exponential in the size of the grammar defining L (cf. Example 4.8). Consequently, the lower bound for computing the reduced core is exponential. We remark, however, that the reduced decomposition can be computed in time polynomial in $|\mathit{Left}(L)| + |\mathit{Right}(L)|$. Also, it is known that the words $\mathit{Left}(L)$ and $\mathit{Right}(L)$ can be represented with singleton grammars of size polynomial in the size of the grammar defining L (cf. [18]), however, it remains to be seen if they can be constructed in polynomial time.

4.2. 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 context free languages, it is constructive.

The result of pushing a word w through a language L will consist of three words: $\mathit{push}(L, w)$ the longest part of w that can be pushed through L, $\mathit{rest}(L, w)$ the part that cannot be pushed through, and $\mathit{offset}(L, w)$ 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 $L = \{\varepsilon\}$. This is the range of a deleting state e.g., $L_{q_3} = \mathit{ran}([\![M_1]\!]_{q_3})$, the range of the state q_3 of M_2 in Example 3.2. This language allows every word to be pushed through and it never changes in the process. For instance, if $w_0 = ab$, then $\mathit{push}(L_{q_3}, w_0) = ab$, $\mathit{rest}(L_{q_3}, w_0) = \varepsilon$, and $\mathit{offset}(L_{q_3}, w_0) = \varepsilon$.

The second class consists of nontrivial periodic languages, essentially languages contained in the Kleene closure of some period word. An example is $L_{q_1} = \mathit{ran}([\![M_1]\!]_{q_1}) = (abc)^* = \{\varepsilon, abc, abcabc, \ldots\}$ 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 $\mathit{push}(L_{q_1}, w_1) = abcabcab$ and $\mathit{rest}(L_{q_1}, w_1) = a$. The offset here is the corresponding prefix of the period: $\mathit{offset}(L_{q_1}, w_1) = ab$.

The third class contains all remaining languages i.e., nontrivial 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 $L_{q_1} = \mathit{ran}([\![M_2]\!]_{q_1}) = \{\varepsilon, a, aab\}$ from Example 3.2, aa is the longest word that can be pushed through. If we take $w_2 = ab$, then we get $\mathit{push}(L_{q_1}, w_2) = a$ and $\mathit{rest}(L_{q_1}, w_2) = b$. Here, the offset is the prefix of aa that has been already pushed through: $\mathit{offset}(L_{q_1}, w_2) = a$. Note that this class also contains the languages that do not permit any pushing through e.g., $L_{q_0} = \mathit{ran}([\![M_2]\!]_{q_0}) = \{ba, ab, aab\}$ 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:

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

For instance, $\mathit{Shovel}(L_{p_1}) = \{\varepsilon, a, aa\}$ and $\mathit{Shovel}(L_{q_0}) = (abc)^* \cdot \{\varepsilon, a, ab\}$. We note that $\mathit{Shovel}(\{\varepsilon\}) = \Delta^*$ and $\mathit{Shovel}(L)$ always contains at least one element ε because L is assumed to be nonempty. Also, if $\mathit{Shovel}(L)$ contains a nonempty word, then L contains the empty word. Indeed, if $a \cdot w \in \mathit{Shovel}(L)$, then a is the first letter of every nonempty word in L, and since L is reduced, L must contain the empty word, or otherwise $\mathit{lcp}(L)$ would be different from ε. Another significant observation follows.

Lemma 4.1 *If L is reduced and nontrivial, then $\mathit{Shovel}(L)$ is prefix-closed and totally ordered by the prefix relation.*

Prefix-closedness follow from the definition. The fact that $\mathit{Shovel}(L)$ is ordered can be proved with a simple induction over the length of words in $\mathit{Shovel}(L)$.

Next, we define periodic languages (cf. [21]). 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 [21] that every nontrivial periodic language L has a unique primitive period, which we denote $\mathit{Period}(L)$. 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 $\mathit{Prefix}(w)$ we denote the set of prefixes of the word w. Below we present an alternative characterization of periodic languages which is useful later on.

Lemma 4.2 *A language L is periodic iff any pair of its words commute i.e., $w_1 \cdot w_2 = w_2 \cdot w_1$ for every $w_1, w_2 \in L$.*

The result above is a direct consequence of know facts about periodic languages [21, Proposition 1.3.2]. Next, we point out an important characterization of periodic languages in terms of the sets of words we can push through them.

Lemma 4.3 *Given a reduced and nontrivial language L, $\mathit{Shovel}(L)$ is infinite iff L is periodic. Furthermore, if L is periodic then $\mathit{Shovel}(L) = \mathit{Period}(L)^* \cdot \mathit{Prefix}(\mathit{Period}(L))$.*

This result and the observations beforehand lead to three relevant cases in the characterization of $\mathit{Shovel}(L)$ for a language L.

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

Now, suppose we wish to *push* a word $w \in \Delta^*$ *through* a reduced language $L \subseteq \Delta^*$. Let s be the longest prefix of w that is present in $\mathit{Shovel}(L)$ and r the reminder i.e., $w = s \cdot r$. Note that because $\mathit{Shovel}(L)$ is closed under prefix, the definition of s is unambiguous. Now, we define $\mathit{push}(L, w)$, $\mathit{rest}(L, w)$, and $\mathit{offset}(L, w)$ depending on the class L belongs to:

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

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

Lemma 4.4 *The set* $\mathit{Offsets}(L) = \{\mathit{offset}(L, w) \mid w \in \Delta^*\}$ *is finite for any reduced* L*.*

The set $\mathit{Offsets}(L)$ can actually be generated by a context-free grammar from a specific word that we denote $\mathit{Kernel}(L)$. When $L = \varepsilon$, $\mathit{Kernel}(L) = \mathit{Offsets}(L) = \varepsilon$. When L is periodic, then $\mathit{Kernel}(L) = \mathit{Period}(L)$ and $\mathit{Offsets}(L) = \mathit{Kernel}(L)^* \mathit{Prefix}(\mathit{Kernel}(L))$. And for other case, we take $\mathit{Kernel}(L)$ as the longest word of $\mathit{Shovel}(L)$ and $\mathit{Offsets}(L) = \mathit{Prefix}(\mathit{Kernel}(L)$. The size of $\mathit{Offsets}(L)$ is at most doubly-exponential in the size of the context-free grammar obtained, and $\mathit{Offsets}(L)$ can be constructed in time polynomial in its size.

4.3. Pushing words backwards

Until now, we have considered the problem of pushing a word through a language from right to left. However, in Example 3.2 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: $w^{-1} \cdot w = \varepsilon$ and $w \cdot w^{-1} = \varepsilon$ for $w \in \Delta^*$, and $v^{-1} \cdot u^{-1} = (uv)^{-1}$ for $u, v \in \Delta^*$. In the sequel, we use $w, u, v, \ldots$ to range over Δ^*

only and $z, z_1, \ldots$ to range over elements of $\mathbb{G}_\Delta$. 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., $ab \cdot a^{-1}$. Furthermore, some expressions may be undefined unless we employ axioms properly. For instance, $ab \cdot (cb)^{-1} \cdot cd$ may be evaluated as $ab \cdot [(cb)^{-1} \cdot cd] = ab \cdot [b^{-1} \cdot d]$ which is undefined but when evaluated diligently we get $ab \cdot b^{-1} \cdot c^{-1} \cdot cd = ad$.

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, $w \cdot L = (u \cdot v) \cdot (v^{-1} \cdot L' \cdot v) = u \cdot L' \cdot v$.

To define pushing backwards more properly we use yet another view 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 we define

$$\begin{aligned} \mathit{push}(L, w^{-1}) &= (\mathit{push}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}, \\ \mathit{rest}(L, w^{-1}) &= (\mathit{rest}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}, \\ \mathit{offset}(L, w^{-1}) &= (\mathit{offset}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}. \end{aligned}$$

The main equation of pushing words through languages is: for every nonempty 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 4.5 *Given a reduced and nonempty language $L \subseteq \Delta^*$ and $z \in \mathbb{G}_\Delta$, for any word $u \in L$*

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

4.4. Normalization algorithm

We can now describe the whole normalization algorithm. We fix an STW $M = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ and first introduce the following macros:

$$L_q = \mathit{ran}([\![M]\!]_q), \quad L_q^\circ = \mathit{Core}(L_q), \quad \mathit{Left}(q) = \mathit{Left}(L_q), \quad \mathit{Right}(q) = \mathit{Right}(L_q),$$
$$\mathit{push}(q, z) = \mathit{push}(L_q^\circ, z), \quad \mathit{offset}(q, z) = \mathit{offset}(L_q^\circ, z), \quad \mathit{rest}(q, z) = \mathit{rest}(L_q^\circ, z).$$

Also, let $\mathit{Offsets}(q) = \{\mathit{offset}(q, z) \mid z \in \mathbb{G}_\Delta\}$ and note that by Proposition 4.4 it is finite. The constructed STW $M' = (\Sigma, \Delta, Q', \mathit{init}', \delta')$ has the following states

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

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

$$[\![M']\!]_{\langle q,z\rangle}(t) = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot [\![M]\!]_q(t) \cdot \mathit{Right}(q)^{-1} \cdot z$$

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 $\delta(q, f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot u_{k-1} \cdot q_k \cdot u_k$ and any $z \in \mathit{Offsets}(q)$ we introduce a rule $\delta'(\langle q, z\rangle, f) = u'_0 \cdot q'_1 \cdot u'_1 \cdot \ldots \cdot u'_{k-1} \cdot q'_k \cdot u'_k$, where $u'_0, \ldots, u'_k$ and $q'_1, \ldots q'_k$ 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:         $u'_i := \mathit{rest}(p_i, z_i)$
4:         $p'_i := \langle p_i, \mathit{offset}(p_i, z_i)\rangle$
5:         $z_{i-1} := \mathit{Right}(p_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(p_i) \cdot \mathit{push}(p_i, z_i)$
6: $u'_0 := z^{-1} \cdot \mathit{Left}(p)^{-1} \cdot z_0$
```

where (for convenience of the presentation) we let $\mathit{Right}(q_0) = \varepsilon$. The complete STW normalization algorithm is presented below.

Algorithm 1 STW normalization.

```
program normalize($M$)
1:  let $M = (\Sigma, \Delta, Q, \mathit{init}, \delta)$
2:  let $\mathit{init} = u_0 \cdot q_0 \cdot u_1$
3:  $Q' := \{\langle q, z\rangle \mid q \in Q,\ z \in \mathit{Offsets}(q)\}$
4:  $v := \mathit{Right}(q_0) \cdot u_1$
5:  $q'_0 := \langle q_0, \mathit{offset}(q_0, v)\rangle$
6:  $u'_0 := u_0 \cdot \mathit{Left}(q_0) \cdot \mathit{push}(q_0, v)$
7:  $u'_1 := \mathit{rest}(q_0, v)$
8:  $\mathit{init}' := u'_0 \cdot q'_0 \cdot u'_1$
9:  for $q \in Q$, $\delta(q, f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot u_{k-1} \cdot q_k \cdot u_k$, and $z \in \mathit{Offsets}(q)$ do
10:     $z_k := \mathit{Right}(q_k) \cdot u_k \cdot \mathit{Right}(q)^{-1} \cdot z$
11:     for $i := k, \ldots, 1$ do
12:         $u'_i := \mathit{rest}(q_i, z_i)$
13:         $q'_i := \langle q_i, \mathit{offset}(q_i, z_i)\rangle$
14:         $z_{i-1} := \mathit{Right}(q_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(q_i) \cdot \mathit{push}(q_i, z_i)$
15:     $u'_0 := z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot z_0$
16:     $\delta'(\langle q, z\rangle, f) = u'_0 \cdot q'_1 \cdot u'_1 \cdot \ldots \cdot u'_{k-1} \cdot q'_k \cdot u'_k$
17: end for
18: $M' := (\Sigma, \Delta, Q', \mathit{init}', \delta')$
19: return $M'$
end program
```

We remark that not all states in Q' need to be 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 4.6 We normalize the STW M_1 from Example 3.2. 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 add a^{-1} at the beginning and a at the end of its right hand side: $a^{-1} \cdot \varepsilon \cdot a = \varepsilon$, 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, so $\mathit{push}(q_1, abca) = abca$, $\mathit{rest}(q_1, abca) = \varepsilon$, and $\mathit{offset}(q_1, abca) = a$. 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 $\delta'(\langle q_1, c^{-1}\rangle, a) = \varepsilon$. $\square$

Theorem 4.7 *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*.*

To prove this theorem, one essentially needs to prove four points:

- the set of states of M' is finite: this is a direct consequence of lemma 4.4
- the languages of the states are correct, technically, For every $q \in Q$, every $z \in \mathit{Offsets}(q)$, and every $t \in \mathit{dom}([\![M]\!]_q)$

$$[\![M']\!]_{\langle q,z\rangle}(t) = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot [\![M]\!]_q(t) \cdot \mathit{Right}(q)^{-1} \cdot z \qquad (2)$$

  This can be proved by induction on the structure of terms t
- $\mathbf{(E_1)}$ and $\mathbf{(E_2)}$ is satisfied in M'. This essentially comes from the fact that $L_{\langle q,z\rangle}$ are reduced
- and finally, M is equivalent to M', i.e. for each t, $u_0[\![M]\!]_q(t)u_1 = u'_0[\![M']\!]_q(t)u'_1$.

A complete proof can be found in the appendix.

4.5. Exponential lower bound

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

Example 4.8 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 4.9 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 $0 \leq i < n$):

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

While the size of this transducer is exponential in n, one can easily compress the exponential factors a^{2^i} and obtain an STW of size linear in n (cf. Example 4.8). M_n satisfies $\mathbf{(E_1)}$ but it violates $\mathbf{(E_2)}$, and defines the following transformation.

$$\begin{aligned} [\![N_n]\!] = \{&(g_{b_0}(g_{b_1}(\ldots g_{b_{n-1}}(a_0)\ldots)), a^{\mathbf{b}}) \mid \mathbf{b} = (b_{n-1}, \ldots, b_0)_2\} \cup \\ \{&(g_{b_0}(g_{b_1}(\ldots g_{b_{n-1}}(a_1)\ldots)), a^{2^n} \cdot \# \cdot a^{\mathbf{b}}) \mid \mathbf{b} = (b_{n-1}, \ldots, b_0)_2\}, \end{aligned}$$

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:

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

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 recognizes $[\![N_n]\!]$. □

5. Myhill-Nerode Characterization and Minimization

The final step in constructing a canonical eSTW representative of a transformation defined by an arbitrary STW is minimization of the normalized eSTW obtained with the procedure detailed in the previous section. The connection between minimization and the existence of canonical representative is generally established with the help of a Myhill-Nerode theorem, which has a number of other interesting applications, including inference which we present in Section 6. In this section we present a Myhill-Nerode characterization for the class of transformation definable with STWs and show how to use this development to devise a polynomial procedure for minimizing eSTWs. Together with the normalization procedure this gives us an effective method for constructing a canonical representative for any transformation definable with an arbitrary STW.

Because STWs process the input tree in a top-down fashion, we decompose τ into several transformations that capture the transformation performed by τ on the children of root of the input tree. This (one-step) 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 then used to define in the standard way the Myhill-Nerode equivalence relation on paths and the canonical transducer.

5.1. Decompositions

The residual construction is based on the notion of decomposition of the transformation, which essentially attempts to express the transformation with a simple production rule (involving factor words and other transformations). First, analogously to defining reduced languages in the previous section, we define reduced transformations: τ is *reduced* iff $lcp(ran(\tau)) = \varepsilon$ and $lcs(ran(\tau)) = \varepsilon$. While not every transformation is reduced, we can reduce any transformation while preserving its essence. Given a transformation τ we define its *core*, denoted $Core(\tau)$, as follows:

$$Left(\tau) = lcp(ran(\tau)), \qquad Right(\tau) = lcs(Left(\tau)^{-1} \cdot ran(\tau)),$$
$$Core(\tau) = \{(t, Left(\tau)^{-1}\tau(t) \cdot Right(\tau)^{-1}) \mid (t, w) \in \tau\}.$$

Example 5.1 Take the transformation $\tau_1(f(g^m(a), g^n(a))) = (abc)^m ac(abc)^n$ for $n, m \geq 0$ (defined by M_1 from Example 3.2). It is not reduced because $Left(\tau_1) = a$ and $Right(\tau_1) = c$. Its core is $\tau_1^\circ = Core(\tau_1)$ with $\tau_1^\circ(f(g^m(a), g^n(a))) = (bca)^m(cba)^n$ for $n, m \geq 0$. Clearly, τ_1^o is reduced, and furthermore, $\tau_1(t) = a \cdot \tau_1^o(t) \cdot c$. □

We state this simple observation in the following.

Proposition 5.2 *For every transformation τ its core $\tau^o = Core(\tau)$ is reduced, $dom(\tau) = dom(\tau^o)$, and furthermore, $\tau(t) = Left(\tau) \cdot \tau^o(t) \cdot Right(\tau)$ for every $t \in dom(\tau)$.*

Now, we define the decomposition of a transformation at a given symbol, which essentially expresses the transformation using a number of (factor) words and (residual) transformations.

Definition 5.3 Given a reduced transformation τ and a symbol $f^{(k)} \in npaths(dom(\tau))$, a *decomposition* of τ for f is a tuple $(u_0, \tau_1, u_1, \ldots, \tau_k, u_k)$, where u_i's are words over Δ and τ_i's are transformations, such that:

(D$_1$) $f(t_1, \ldots, t_k) \in dom(\tau)$ if and only if $t_i \in dom(\tau_i)$ for all $1 \leq i \leq k$ and for all trees $t_1, \ldots, t_k$;

($\mathbf{D_2}$) $\tau(f(t_1,\dots,t_k)) = u_0 \cdot \tau_1(t_1) \cdot \dots \cdot \tau(t_k) \cdot u_k$ for all $f(t_1,\dots,t_k) \in \mathit{dom}(\tau)$;
($\mathbf{C_1}$) τ_i is reduced for every $1 \leq i \leq k$;
($\mathbf{C_2}$) $\mathit{lcp}(\mathit{ran}(\tau_i) \cdot u_i \cdot \dots \cdot \mathit{ran}(\tau_k) \cdot u_k) = \varepsilon$ for every $1 \leq i \leq k$.

The conditions **($\mathbf{D_1}$)** and **($\mathbf{D_2}$)** ensure that the original transformation can be precisely reproduced from the decomposition, and the conditions **($\mathbf{C_1}$)** and **($\mathbf{C_2}$)** are reformulations of **($\mathbf{E_1}$)** and **($\mathbf{E_2}$)** that ensure uniqueness of decompositions. We observe, however, that a transformation needs not have a decomposition for every symbol as shown in the following example.

Example 5.4 Consider the reduced transformation τ given with the following equations

$$\tau(f(a,a)) = abc, \quad \tau(f(a,b)) = abac, \quad \tau(f(b,a)) = aabc, \quad \tau(f(b,b)) = aabac,$$
$$\tau(g(a,a)) = \varepsilon, \quad \tau(g(b,a)) = b, \quad \tau(g(a,b)) = bb, \quad \tau(g(b,b)) = bbb.$$

The decomposition of τ at f is the tuple (a, τ', b, τ', c), where $\tau'(a) = \varepsilon$ and $\tau'(b) = a$ (otherwise τ' is undefined). On the other hand, τ does not have a decomposition at g. □

We can, however, show that if there exists a decomposition then it is unique because the conditions **($\mathbf{C_1}$)** and **($\mathbf{C_2}$)** ensure the canonicity of decomposition.

Lemma 5.5 *For any transformation τ and any $f \in \mathit{npaths}(\mathit{dom}(\tau))$, τ has at most one decomposition for f.*

This claim is proved by contradiction by assuming two different decompositions which leads to a violation of **($\mathbf{C_1}$)** when the decompositions differ on some factor or a violation of **($\mathbf{C_2}$)** when the decomposition differ with the residual transformations. A complete proof can be found in appendix.

The analogy between the conditions of **($\mathbf{C_1}$)** and **($\mathbf{C_2}$)** and the conditions **($\mathbf{E_1}$)** and **($\mathbf{E_2}$)** allows us to make an important connection between decompositions of rules an eSTWs.

Proposition 5.6 *For an eSTW M, any state q of M, $[\![M]\!]_q$ is reduced, and furthermore for any $f \in \Sigma$ such that $\delta(q,f)$ is defined, the decomposition of $[\![M]\!]_q$ for f is $(u_0, [\![M]\!]_{q_1}, u_1, \dots, [\![M]\!]_{q_k}, u_k)$, where $\delta(q,f) = u_0 \cdot q_1 \cdot u_1 \cdot \dots \cdot q_k \cdot u_k$.*

Proof We note that $[\![M]\!]_q$ is reduced by **($\mathbf{E_1}$)**, while **($\mathbf{D_1}$)** and **($\mathbf{D_2}$)** follow directly from the definition of $[\![M]\!]_q$. **($\mathbf{C_1}$)** follows immediately from **($\mathbf{E_1}$)** and **($\mathbf{C_2}$)** from **($\mathbf{E_2}$)**. □

5.2. Residuals

The notion of decomposition plays a central role in the definition of a residual of a transformation.

Definition 5.7 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.

We illustrate this important notion on the example of the transformation τ_1 defined by M_1 in Example 3.2.

Example 5.8 Recall that $\tau_1(f(g^m(a), g^n(a))) = (abc)^m ac(abc)^n$ for $n, m \geq 0$ (Example 3.2) and its core $\tau_1^\circ(f(g^m(a), g^n(a))) = (bca)^m(cab)^n$ for $n, m \geq 0$ (Example 5.1). Naturally, $\epsilon^{-1}\tau_1 = \tau_1^\circ$. The decomposition of τ_1° at f is $(\varepsilon, \tau_{1,1}, \varepsilon, \tau_{1,2}, \varepsilon)$, where $\tau_{1,1}(g^m(a)) = (bca)^m$ for $m \geq 0$ and $\tau_{1,2}(g^n(a)) = (cab)^n$ for $n \geq 0$. Naturally, $(f,1)^{-1}\tau_1 = \tau_{1,1}$ and $(f,2)^{-1}\tau_1 = \tau_{1,2}$. The decomposition of $\tau_{1,1}$ at g is $(bca, \tau_{1,1}, \varepsilon)$ and the decomposition of $\tau_{1,1}$ at a is ε (a is a input symbol of arity 0). Analogously, he decomposition of $\tau_{1,2}$ at g is $(cba, \tau_{1,2}, \varepsilon)$ and the decomposition of $\tau_{1,2}$ at a is ε. Therefore, $[(f,1)\cdot(g,1)^n]^{-1}\tau = \tau_{1,1}$ and $[(f,2)\cdot(g,1)^n]^{-1}\tau = \tau_{1,2}$ for any $n \geq 0$.

Again, we point out an important connection between the residuals and the transformations defined by the states of eSTWs.

Lemma 5.9 *Take an* eSTW M *and let* q_0 *be its initial state. For every* $p \in$ *paths*$(dom(\llbracket M \rrbracket))$ *the residual* $p^{-1}\llbracket M \rrbracket$ *exists and is equal to* $\llbracket M \rrbracket_{\delta(q_0,p)}$.

This is essentially an inductive extension of Proposition 5.6. Its complete prove is in Appendix. Naturally, not every transformation has well-defined residuals. The above result shows, however, that any transformation definable by an STW has them and there is a strict correspondence between the residuals and the states of an equivalent eSTW defining the transformation.

5.3. Myhill-Nerode characterization

In the remainder of this section, we are interested in transformations that have a residual for every path of their domain.

Definition 5.10 A tree-to-word transformation τ is *sequential top-down* if $p^{-1}\tau$ exists for every $p \in dom(\tau)$. By $\mathcal{STW}$ we denote the class of all sequential top-down transformations.

Having defined residuals, the construction of the canonical transducer $Can(\tau)$ 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 $p_1, p_2 \in paths(dom(\tau))$. The *Myhill-Nerode equivalence class* of a path p w.r.t. τ is $[p]_\tau = \{p' \in paths(dom(\tau)) \mid p \equiv_\tau p'\}$. We say that τ has *finite Myhill-Nerode index* if $\equiv_\tau$ has a finite number of equivalence classes. The *canonical transducer* of an $\mathcal{STW}$ τ with a finite Myhill-Nerode index is an eSTWs $Can(\tau) = (\Sigma, \Delta, Q, init, \delta)$, whose states are equivalence classes of the Myhill-Nerode equivalence relation $\equiv_\tau$:

$$Q = \{[p]_\tau \mid p \in paths(dom(\tau))\},$$

the initial state is constructed simply as

$$init = Left(\tau) \cdot [\varepsilon]_\tau \cdot Right(\tau).$$

and for $p \cdot f \in \mathit{npaths}(\mathit{dom}(\tau))$ to define the transition rule $\delta([p]_\tau, f)$ we take the decomposition $(u_0, \tau_1, u_1, \ldots, \tau_k, u_k)$ of $p^{-1}\tau$ for f and assign

$$\delta([p]_\tau, f) = u_0 \cdot [p \cdot (f,1)]_\tau \cdot u_1 \cdot \ldots \cdot [p \cdot (f,k)]_\tau \cdot u_k.$$

Note that this assignment is independent of the choice of the representative of the equivalence class because if $p_1 \equiv_\tau p_2$ the residuals $p_1^{-1}\tau$ and $p_2^{-1}\tau$ are the same, and in particular, have the same decompositions.

Theorem 5.11 *A transformation τ is sequential top-down and has a finite Myhill-Nerode index if and only if τ is defined an* eSTW.

PROOF For the *if* part, we observe that if τ is defined by eSTW M, then by Lemma 5.9 τ is sequential top-down and its Myhill-Nerode index is bounded by the number of states of M. For the *only if* part, we show that the canonical transducer of τ indeed defines τ. This is done with a simple induction (over the height of the input tree) showing that that $[\![\mathit{Can}(\tau)]\!]_{[p]_\tau} = p^{-1}\tau$ for every $p \in \mathit{paths}(\mathit{dom}(\tau))$. □

5.4. Minimization

We next show that $\mathit{Can}(\tau)$ is in fact the unique minimal eSTW that recognizes τ. For that, we first need to establish a bisimulation property for pairs of equivalent eSTWs which also allows us to devise a polynomial procedure for minimizing eSTWs.

Lemma 5.12 *Take two* eSTWs *$M = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ and $M' = (\Sigma, \Delta, Q', \mathit{init}', \delta')$ defining the same transformation $\tau = [\![M]\!] = [\![M']\!]$ and let $\mathit{init} = u_0 \cdot q_0 \cdot u_1$ and $\mathit{init}' = u_0' \cdot q_0' \cdot u_1'$. Then, $u_0 = u_0'$ and $u_1 = u_1'$, and for every $p \in \mathit{paths}(\mathit{dom}(T))$, we let $q = \delta(q_0, p)$ and $q' = \delta'(q_0', p)$, and we have*

1. *$[\![M]\!]_q = [\![M']\!]_{q'}$,*
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 $0 \le i \le k$.*

Those three points can be proved conjointly by a recursion on p (a complete proof in Appendix).

For the reminder of this subsection we fix an eSTW $M = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ and let $\mathit{init} = u_0 \cdot q_0 \cdot u_1$ be its initial rule. We define the following equivalence relation on states of M: $q \equiv_M q'$ if and only if $[\![M]\!]_q = [\![M]\!]_{q'}$. This equivalence relation can be precomputed using PTIME equivalence tests for eSTWs [31]. In the sequel, by $[q]_{\equiv_M} = \{q' \in Q \mid q \equiv_M q'\}$ we denote the equivalence class of state q w.r.t. $\equiv_M$. The result of minimization is the *quotient transducer* $M/_{\equiv_M} = (\Sigma, \Delta, Q', \mathit{init}', \delta')$, where $Q' = \{[q]_{\equiv_M} \mid q \in Q\}$, $\mathit{init}' = u_0 \cdot [q]_{\equiv_M} \cdot c_1$, $\delta([q]_{\equiv_M}, f) = u_0 \cdot [q_1]_{\equiv_M} \cdot u_1 \cdot \ldots \cdot [q_k]_{\equiv_M} \cdot u_k$ for any rule $\delta(q, f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot q_k \cdot u_k$ of M. Note that Lemma 5.12 (with $M' = M$) guarantees that the construction of a rules of $M/_{\equiv_M}$ is independent on the choice of the representative rule of M.

Lemma 5.13 *$M/_{\equiv_M}$ is the minimal unique* eSTW *defining $[\![M]\!]$.*

The proof of this lemma uses the standard minimization arguments that are enabled by the conditions $\mathbf{(E_1)}$ and $\mathbf{(E_2)}$ (complete proof in Appendix). As a consequence we obtain an important result.

Theorem 5.14 *Minimization of* eSTWs *is in* PTIME *i.e, for every* eSTW *there exists a unique minimal equivalent* eSTW *that can be constructed in polynomial time.*

The developments presented in this section have a broader impact if we observe the observation that the equivalence relation $\equiv_M$ is close representation of the Myhill-Nerode equivalence relation $\equiv_{[\![M]\!]}$, which can be stated as

$$\forall p_1, p_2 \in paths(dom([\![M]\!])).\ \delta(q_0, p_1) \equiv_M \delta(q_0, p_2) \quad \text{iff} \quad p_1 \equiv_{[\![M]\!]} p_2.$$

which shows that the quotient transducer is in fact the canonical transducer (complete proof in Appendix).

Lemma 5.15 *For any* eSTW *M, $Can([\![M]\!]) = M/_{\equiv_M}$.*

We summarize the results obtained in the following corollary.

Corollary 5.16 (Myhill-Nerode characterization for stw) *For any tree-to-word transformation τ the following three 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.*

We finish by pointing out that the conditions $\mathbf{(E_1)}$ and $\mathbf{(E_2)}$ play an important role in enabling a tractable minimization because for arbitrary STWs the output words may be arbitrarily distributed among the rules, which is a major challenge and is unlikely to be easy to overcome as suggested by the following result.

Theorem 5.17 *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.*

This is proved using a reduction of the problem $\texttt{SAT}_{\texttt{ONE-IN-THREE}}$, known to be NP-complete (complete proof in Appendix).

6. Learning STWs

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

6.1. 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 [14]. 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 6.1 *Checking if there exists an* STW *consistent with a given sample is NP-complete.*

Proof of this theorem is in the appendix. 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 [24].

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 6.2 $\mathcal{STW}$ *transformations represented by* eSTW *are learnable with abstain from polynomial time and data.*

6.2. 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 $dom(\tau)$.

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 $\mathit{paths}(\mathit{dom}(S))$ contains all smallest paths representative of each residual, this procedure produces exactly the set Q of states of $\mathit{Can}(\tau)$. 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.

Algorithm 2 $\mathsf{learner}_D(S)$

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

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 6.3 *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 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 $u_0 \cdot S_1 \cdot u_1 \ldots S_k \cdot u_k$ which ideally is the proper decomposition of S w.r.t. to τ.

Algorithm 3 $\mathsf{decomp}(S, f^{(k)})$

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

6.3. 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 $\mathit{StatePath}(\tau) = \{\min_{\mathit{Path}}([p]_\tau) \mid p \in \mathit{paths}(\mathit{dom}(\tau))\}$. We also define $\mathit{Kernel}(\tau)$, which adds to the shortest paths their extensions with one additional step i.e., $\mathit{Kernel}(\tau) = \mathit{StatePath}(\tau) \cup \{p \cdot (f, i) \in \mathit{paths}(\mathit{dom}(\tau)) \mid p \in \mathit{StatePath}(\tau)\}$.

Example 6.4 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 $\Delta = \{c\}$ that counts the number of symbols in t (i.e. $\tau_1(f(f(a,b),a)) = ccccc$). The canonical transducer of τ_1 is $M_1 = (\Sigma, \Delta, Q_1 = \{q\}, \mathit{init}_1 = c \cdot q, \delta_1)$ with $\delta_1(q, a) = \varepsilon$, $\delta_1(q, b) = \varepsilon$ and $\delta_1(q, f) = cc \cdot q \cdot q$.

The transformation τ_1 has 3 distincts residuals: $\varepsilon^{-1}\tau_1$, $(f,1)^{-1}\tau_1$ and $(f,2)^{-1}\tau_1$. Therefore, $\mathit{StatePath}(\tau_1) = \{\varepsilon, (f,1), (f,2)\}$ and $\mathit{Kernel}(\tau_1) = \mathit{StatePath}(\tau_1) \cup \{(f,1)(f,1),\ (f,1)(f,2),\ (f,2)(f,1),\ (f,2)(f,2)\}$.

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

($\mathbf{S_0}$) the tree $\min_{\mathit{Tree}}(\mathit{dom}(p^{-1}\tau))$ belongs to T;

($\mathbf{S_1}$) $\mathit{lcp}((p^{-1}\tau)(T)) = \varepsilon$ and $\mathit{lcs}((p^{-1}\tau)(T)) = \varepsilon$;

($\mathbf{S_2}$) $\mathit{lcp}(\mathit{ran}(p^{-1}\tau) \setminus \{\varepsilon\}) = \mathit{lcp}((p^{-1}\tau)(T) \setminus \{\varepsilon\})$.

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

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

For a path p, conditions **($\mathbf{S_0}$)**, **($\mathbf{S_1}$)** and **($\mathbf{S_2}$)** 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 6.5 Consider transformation τ_1 of example 6.4 and take for instance $p = (f, 1)$. The tree $T_{p,\varepsilon}$ is the smallest tree whose image differs in $p^{-1}\tau_1$ and $\varepsilon^{-1}\tau_1$. In fact, $T_{p,\varepsilon} = f(a, a)$ as $p^{-1}\tau_1(f(a, a)) = cc$ and $\varepsilon^{-1}\tau_1(f(a, a)) = ccc$. For other $p' \in \{(f, 2), (f, 2)(f, 1), (f, 2)(f, 2)\}$, $T_{p,p'} = a$.

To satisfy condition $(\mathbf{S_1})$ and $(\mathbf{S_2})$, one can take $\{a, b, f(a, a)\} \in T_p$. This allows to satisfy $(\mathbf{S_1})$ as $lcp(\{(p^{-1}\tau_1)(a), (p^{-1}\tau_1)(b), (p^{-1}\tau_1)(f(a, a))\}) = lcp(\{c,$ ε, $ccc\}) = \varepsilon$ and the same for lcs. For $(\mathbf{S_2})$, we have $lcp(\{(p^{-1}\tau_1)(a)$, $(p^{-1}\tau_1)(b)$, $(p^{-1}\tau_1)(f(a, a))\} \setminus \{\varepsilon\}) = lcp(\{c, \varepsilon, ccc\} \setminus \{\varepsilon\}) = c$ which is indeed equal to $lcp(ran(p^{-1}\tau_1) \setminus \{\varepsilon\})$.

Let τ be a transformation in $\mathcal{STW}$ and let p be a path in $Kernel(\tau)$. 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 $p \cdot p_0 \in Kernel(\tau)$, 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 $(\mathbf{S_0})$, $(\mathbf{S_1})$, and $(\mathbf{S_2})$ 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 6.6 *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 6.7 From previous example, one can build a characteristic sample for τ_1. In particular, the minimal context for $(f, 1)$ is $f(x, a)$. In example 6.5, it is argued that trees $\{a, b, f(a, a)\}$ are in T_p, which means that $CharSet(\tau_1)$ contains $(f(a, a), ccc)$, $(f(b, a), cc)$ $(f(f(a, a), a), ccccc)$. A similar approach has to be also considered for all other elements of $Kernel(\tau_1)$ to obtain the full $CharSet(\tau_1)$.

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 6.8 *Let* $\tau \in \mathcal{STW}$ *and* $p \in StatePath(\tau)$. *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* $u_0 \cdot \tau_1 \ldots \tau_k \cdot u_k$, *then* $\mathsf{decomp}(S, f) = u_0 \cdot S_1 \ldots S_k \cdot u_k$ *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 $lcp(W \setminus \{\varepsilon\}) = lcp(W' \setminus \{\varepsilon\})$, and $lcp(W) = lcp(W')$, then $lcp(\{w \cdot u \mid w \in W\}) = \varepsilon$ for a $u \in \Delta^*$ implies that $lcp(\{w' \cdot u \mid w' \in W'\}) = \varepsilon$.

Now, consider a transformation $\tau \in \mathcal{STW}$, a path $p \in StatePath(\tau)$ 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 $lcp\{\tau_i(t_i) \cdot u_i \cdot \dots \cdot \tau_k(t_k) \cdot u_k \mid t_i \in (f,i)^{-1}dom(S), \dots, t_k \in (f,k)^{-1}dom(S)\} = \varepsilon$.. 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 6.8.

As the construction of residuals $\mathsf{residual}(S,p)$ relies on the decomposition, Lemma 6.8 has the important consequence that those residuals can be computed properly for any $p \in Kernel(\tau)$. This gives the following two results. First, if S is characteristic for τ, and $p \in Kernel(\tau)$, then $\mathsf{residual}(S,p)$ is characteristic for τ w.r.t. p. Second, as a consequence and because of $\mathbf{(DI)}$, if $p, p' \in Kernel(\tau)$ 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 $Can(\tau)$:

Lemma 6.9 *Let $\tau \in \mathcal{STW}$ and D a* DTA *with $[\![D]\!] = dom(\tau)$. From any sample S characteristic with τ, $\mathsf{learner}_D(S) = Can(\tau)$.*

This, along with Lemmas 6.3 and 6.6 proves Theorem 6.2.

7. Conclusion

We have proposed and studied the class of deterministic top-down sequential tree-to-word transducers (STWs). The core contribution of this paper, and virtually the source of all contributions of this paper, is a normal form for STWs: we have identified two syntactic conditions that yield a proper subclass of *earliest* STWs. The conditions are easily to verify and furthermore do not restrict the expressive power the transducers: we show that for an STW an equivalent eSTW can be constructed. One important ramification of the normal form for STWs is that when adapted to tree-to-word transformations (semantic rather than syntactic object), it yields a Myhill-Nerode characterization of transformations definable with STWs that allows to identify a unique canonical eSTW representative of any transformation from this class. This connection becomes clear when we observe that the conditions $\mathbf{(C_1)}$ and $\mathbf{(C_2)}$ in Definition 5.3 that ensure unique canonical eSTW are essentially reformulations of the conditions $\mathbf{(E_1)}$ and $\mathbf{(E_2)}$ in Definition 3.4 (that defined the normal form for STWs). Naturally, we also provide a (polynomial) minimization algorithm for eSTWs, which together yields an effective procedure for converting any STW into its unique canonical eSTW representative. The Myhill-Nerode characterization has a number of other uses and applications, and in this paper, we use it to devise a learning algorithm for the class of earliest STWs. To the best of our knowledge this is the first learning algorithm allowing to infer tree-to-word transducers.

We envision a number of possible directions of further study. First, we would like to adapt the results to more generalized models of top-down tree-to-word transducers. Our preliminary results suggest that allowing arbitrary order (a permutation) in which the subtrees are processed by the transducer may still allow for an analogous normal form, albeit slightly more developed. Allowing the transducer to visit a node more than posses, however, a more significant challenge which we are yet to see if it can be overcome with techniques analogous to those presented in this paper. This naturally leads us to the open problem of normalizing the general class of Macro Tree Transducers. Finally, we would like to explore studing learning subclasses of STWs under alternative learning frameworks, in particular approximate and probabilistic learning frameworks. It should be noted, however, that intractability of the consistency problem for a given class of concepts makes it generally difficult to render it learnable in the popular framework of PAC-learnability [32].

[1] Rajeev Alur, Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Congruences for visibly pushdown languages. In *Automata, Languages and Programming, 32nd International Colloquium*, volume 3580 of *Lecture Notes in Computer Science*, pages 1102–1114. Springer Verlag, 2005.

[2] Jean Berstel. *Transductions and Context-Free Languages.* Teubner Studienbucher, 1979.

[3] 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.

[4] G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems (TODS)*, 35(2), 2010.

[5] Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning rational functions. In Hsu C. Yen, Oscar H. Ibarra, Hsu C. Yen, and Oscar H. Ibarra, editors, *Developments in Language Theory*, volume 7410 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 2012.

[6] J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1):33–67, 2007.

[7] Christian Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131–143, 2003.

[8] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available online since 1997: `http://tata.gforge.inria.fr`, October 2007.

[9] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38:1332–1348, September 2005.

[10] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R. V. Book, editor, *Formal language theory; perspectives and open problems*. Academic Press, New York, 1980.

[11] 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.

[12] Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Properties of visibly pushdown transducers. In *35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10)*, volume 6281 of *Lecture Notes in Computer Science*, pages 355–367. Springer Verlag, 2010.

[13] Sylvia Friese, Helmut Seidl, and Sebastian Maneth. Minimization of deterministic Bottom-Up tree transducers. In Yuan Gao, Hanlin Lu, Shinnosuke Seki, and Sheng Yu, editors, *Developments in Language Theory, 14th International Conference DLT 2010*, volume 6224 of *Lecture Notes in Computer Science*, pages 185–196. Springer Verlag, 2010.

[14] E. M. Gold. Complexity of automaton identification from given data. *Inform. Control*, 37:302–320, 1978.

[15] T. V. Griffiths. The unsolvability of the equivalence problem for Lambda-Free nondeterministic generalized machines. *Journal of the ACM*, 15(3):409–413, 1968.

[16] Eitan M. Gurari. The equivalence problem for deterministic Two-Way sequential transducers is decidable. *SIAM Journal on Computing*, 11(3):448–452, 1982.

[17] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[18] Juhani Karhumäki and Wojciech Plandowski. On the size of independent systems of equations in semigroups. In Igor Pr'ıvara, Branislav Rovan, Peter Ruzicka, Igor Pr'ıvara, Branislav Rovan, and Peter Ruzicka, editors, *MFCS*, volume 841 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 1994.

[19] 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.

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

[21] M. Lothaire, editor. *Combinatorics on Words*. Cambridge Mathematical Library. Cambridge University Press, 2nd edition, 1997.

[22] Sebastian Maneth. The macro tree transducer hierarchy collapses for functions of linear size increase. In P. K. Pandya and J. Radhakrishnan, editors, *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science – FSTTCS'2003*, volume 2914 of *LNCS*, pages 326–337. Springer-Verlag, 2003.

[23] Wim Martens, Frank Neven, and Marc Gyssens. Typechecking top-down XML transformations: Fixed input or output schemas. *Inf. Comput.*, 206(7):806–827, 2008.

[24] J. Oncina and P. García. Inference of recognizable tree sets. Technical report, Departamento de Sistemas Informáticos y Computación, Universidad de Alicante, 1993. DSIC-II/47/93.

[25] 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.

[26] J. Oncina and P. Gracia. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, pages 99–108, 1992.

[27] Jean-François Raskin and Frédéric Servais. Visibly pushdown transducers. In *Automata, Languages and Programming, 35th International Colloquium*, volume 5126 of *Lecture Notes in Computer Science*, pages 386–397. Springer Verlag, 2008.

[28] C. Reutenauer and M. P. Schützenberger. Minimalization of rational word functions. *SIAM Journal on Computing*, 20:669–685, 1991.

[29] Thomas J. Schaefer. The complexity of satisfiability problems. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, New York, NY, USA, 1978. ACM.

[30] S. Staworko and P. Wieczorek. Learning twig and path queries. In *International Conference on Database Theory (ICDT)*, March 2012.

[31] Slawek Staworko, Grégoire Laurence, Aurélien Lemay, and Joachim Niehren. Equivalence of nested word to word transducers. In *17th International Symposium on Fundamentals of Computer Theory*, volume 5699 of *Lecture Notes in Computer Science*, pages 310–322. Springer Verlag, 2009.

[32] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.

A. Canonical orders

We give here a precise definition of an example of canonical well-founded orders that can be used on trees, words and paths. First, we let $\Sigma_\Diamond = \Sigma \times \{\mathit{open}, \mathit{close}\}$ and define the *preorder traversal* of a tree t over Σ as a word over $\Sigma_\Diamond$ obtained in a recursive fashion:

$$\mathit{traverse}(f(t_1, \dots, t_k)) = (f, \mathit{open}) \cdot \mathit{traverse}(t_1) \cdot \ldots \cdot \mathit{traverse}(t_k) \cdot (f, \mathit{close}).$$

We extend the notion of traversal to context in the natural fashion (by adding $x^{(0)}$ to the alphabet).

We recall next a general procedure for extending a total order $\leq_A$ on A to a canonical well-founded order $\leq_A^{\mathrm{can}}$ on words over A. The *lexicographical* order $\leq_A^{\mathrm{lex}}$ on A^* induced by $\leq_A$ is defined as follows: $w \leq_A^{\mathrm{lex}} u$ iff w is a prefix of u or $w = v \cdot a \cdot z$, $u = v \cdot b \cdot z'$, and $a \leq_A b$, where $u, w, v, z, z' \in A^*$ and $a, b \in A$. We observe that lexicographic orders are generally not well-founded e.g., if $a \leq b$, then $\ldots \leq^{\mathrm{lex}} aab \leq^{\mathrm{lex}} ab \leq^{\mathrm{lex}} b$. Consequently, to obtain a *canonical well-founded order* we order the words by their length and then order the words of the same length with the use of the lexicographical order i.e., $u \leq_A^{\mathrm{can}} w$ iff $|u| \leq |w|$ or $|u| = |w|$ and $u \leq_A^{\mathrm{lex}} w$. In our constructions, we also employ the (*lexicographic*) *composition* of two total orders $\leq_A$ on A and $\leq_B$ on B to obtain a total order $\leq_{A\times B}$ on $A \times B$ defined as $(a_1, b_1) \leq_{A\times B} (a_2, b_2)$ iff $a_1 \leq_A a_2$ or $a_1 = a_2$ and $b_1 \leq_B b_2$.

The basic orders that we use are: 1) An arbitrary total order $\leq_\Sigma$ on Σ that is efficiently testable; 2) The standard total order $\leq$ on natural numbers; 3) the total order $\leq_\Diamond$ on $\{\mathit{open}, \mathit{close}\}$ such that $\mathit{open} \leq_\Diamond \mathit{close}$. As usually, given an order $\leq_A$ on A, when we write $a <_A b$ we mean $a \leq_A b$ and $a \neq b$. Analogously, $a \geq_A b$ is $b \leq_A b$, and $a >_A b$ is $a \geq_A b$ and $a \neq b$.

Now, we compose $\leq_\Sigma$ and $\leq$ (on $\mathbb{N}$) to obtain the total order $\leq_{\Sigma\times\mathbb{N}}$ on $\Sigma \times \mathbb{N}$. Next, we extend $\leq_{\Sigma\times\mathbb{N}}$ to the canonical well-founded order $\leq_{\mathit{Path}}$ on words over $\Sigma \times \mathbb{N}$ i.e., labeled paths.

Next, we take the total ordering $\leq_\Diamond$ on $\{\mathit{open}, \mathit{close}\}$ such that $\mathit{open} \leq_\Diamond \mathit{close}$ and construct the total order $\leq_{\Sigma_\Diamond}$ by the composition of $\leq_\Sigma$ and $\leq_\Diamond$. We extend $\leq_{\Sigma_\Diamond}$ to the canonical order $\leq_{\Sigma_\Diamond}^{\mathrm{can}}$ on words over $\Sigma_\Diamond$ and define the well-founded order on trees induced by preorder traversal: $t_1 \leq_{\mathit{Tree}} t_2$ iff $\mathit{traverse}(t_1) \leq_{\Sigma_\Diamond}^{\mathrm{can}} \mathit{traverse}(t_2)$. Analogously to $\leq_{\mathit{Tree}}$, we define a well-founded order $\leq_{\mathit{Ctx}}$ on contexts: in the construction above we extend $\leq_\Sigma$ to the place-holder $x^{(0)}$ and make $x^{(0)}$ the minimal element of $\leq_\Sigma$, the reminder of the construction remains unchanged.

B. Pushing Words Through Languages

Lemma 4.1 If L is reduced and nontrivial, then $\mathit{Shovel}(L)$ is prefix-closed and totally ordered by the prefix relation.

Proof Showing that $\mathit{Shovel}(L)$ is prefix-closed follows from the definition. Take any $w \in \mathit{Shovel}(L)$ and a prefix w' of w whose length is $|w'| = k$. Now, fix a word $v \in L$ and observe that w is a prefix of $v \cdot w$. Since w' is a prefix of w, then $v \cdot w'$ is a prefix of $v \cdot w$ and furthermore $v \cdot w'$ is of length at least k. Consequently, w' is a prefix of $v \cdot w'$.

We show that $\mathit{Shovel}(L)$ is ordered with a simple induction over the length of words in $\mathit{Shovel}(L)$. Take two words $w \cdot a, w \cdot b \in \mathit{Shovel}(L)$. Since L is nontrivial, there is some nonempty word $u \in L$. Now, $w \cdot a$ is a prefix of $u \cdot w \cdot a$ and $w \cdot b$ is a prefix of $u \cdot w \cdot b$. Since u has length at least 1, both $w \cdot a$ and $w \cdot b$ are prefixes of $u \cdot w$. Consequently, $a = b$. □

Lemma 4.3 Given a reduced and nontrivial language L, $\mathit{Shovel}(L)$ is infinite iff L is periodic. Furthermore, if L is periodic then $\mathit{Shovel}(L) = \mathit{Period}(L)^* \cdot \mathit{Prefix}(\mathit{Period}(L))$.

Proof For the *if* part, take any nontrivial $L \subseteq w^*$ and observe that $w^k \in \mathit{Shovel}(L)$ for any $k \geq 0$. Furthermore, by Proposition 4.1 we get that $\mathit{Shovel}(L) = w^* \cdot \mathit{Prefix}(w)$.

For the *only if* part we point out that Proposition 4.2 characterizes nontrivial periodic languages as exactly those that self-commute i.e., a nontrivial L is periodic iff $w_1 \cdot w_2 = w_2 \cdot w_1$ for any two words $w_1, w_2 \in L$.

First, we observe that $\varepsilon \in L$. Since $\mathit{Shovel}(L)$ contains a nonempty word $a \cdot w \in \mathit{Shovel}(L)$, then a is the first letter of every nonempty word in L, and since L is trimmed, L must contain the empty word, or otherwise $\mathit{lcp}(L)$ would not be ε.

Next, take any $w_1, w_2 \in L$. Since $\mathit{Shovel}(L)$ is infinite, there exists a word $v \in \mathit{Shovel}(L)$ whose length is greater than $|w_1| + |w_2|$. In addition we remark that v is a prefix of v, $w_1 \cdot v$, and $w_2 \cdot v$. This allows us to infer that $v = w_1 \cdot v'$ and $v = w_2 \cdot v''$, which implies that $w_1 \cdot v = w_1 \cdot w_2 \cdot v''$ and $w_2 \cdot v = w_2 \cdot w_1 \cdot v'$. Since the length of v is greater than $|w_1| + |w_2|$, the aforementioned two words agree on the first $|w_1| + |w_2|$ letters and hence $w_1 \cdot w_2 = w_2 \cdot w_1$. □

Lemma 4.4 The set $\mathit{Offsets}(L) = \{\mathit{offset}(L, w) \mid w \in \Delta^*\}$ is finite for any reduced L. Furthermore, if L is defined by a context-free grammar G, then the size of $\mathit{Offsets}(L)$ is at most doubly-exponential in the size of G and $\mathit{Offsets}(L)$ can be constructed in time polynomial in its size.

Proof The fact that the set $\mathit{Offsets}(L)$ is finite follows directly from the definitions in all three cases. We just show that this set can be constructed in time doubly-exponential in the size of a context-free grammar G defining L. The algorithms outlined below allow also to classify the case the language L belongs to.

For the case 0^o we note that the condition $L = \{\varepsilon\}$ can be easily checked on G by testing that every (reachable) nonterminal of G does not produce the non-empty word. This can be done with a simple closure algorithm working in time polynomial in the size of G.

We handle the remaining two cases together and let $w_{\min}$ a shortest word in L. First, we observe that L is periodic if and only if its primitive period $\mathit{Period}(L)$ is also the primitive period of $w_{\min}$. Now, let v be the shortest prefix

v of $w_{\min}$ such that $w_{\min} = v^k$ for some $k > 0$ (possibly equal to $w_{\min}$). One can easily see that $\mathit{Period}(\{w_{\min}\})$ is the shortest prefix v of $w_{\min}$ such that $w_{\min} = v^k$ for some $k > 0$. Consequently, L is periodic if and only if $L \subseteq v^*$.

We observe that $w_{\min}$ may be of length exponential in the size of G (cf. Example 4.8), and that may be also the length of v. Testing the inclusion $L \subseteq v^*$, when G is a CFG defining L, can be done using standard automata techniques: we construct a push-down automaton A_G defining L, a DFA A_v defining v^*, and its complement $A_v^{\complement}$ defining $\Delta^* \setminus v^*$. Now, we take the product $P = A_G \times A_v^{\complement}$ and test it for emptiness. Clearly, $L \subseteq v^*$ iff P defines an empty language. As for complexity, we note that the size of A_G is $\mathit{poly}(|G|)$, the sizes of A_v and $A_v^{\complement}$ are $\mathit{poly}(|v|) = \mathit{exp}(|G|)$, and thus the product automaton P is of size $\mathit{exp}(|G|)$.

If L is periodic, then $O = \mathit{Prefix}(\mathit{Period}(L)) = \mathit{Prefix}(v)$ and its size is single-exponential in the size of G. If L is not periodic, then the test described above fails i.e., P is nonempty and accepts an non-empty word w_0. Note that because P is a push-down automaton whose size is $\mathit{exp}(|G|)$ the shortest word recognized by P may be of size doubly-exponential in the size of G. We claim that if a word can be pushed through L, then it cannot be longer than w_0. The proof is combinatorial and we omit it here. □

Lemma 4.5 Given a reduced and nonempty language $L \subseteq \Delta^*$ and $z \in \mathbb{G}_\Delta$, for any word $u \in L$

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

Proof We distinguish 3 cases: L is trivial or L is periodic and nontrivial or L is non periodic. If L is trivial, then $\mathit{push}(L, z) = \mathit{rest}(L, z) = \mathit{offset}(L, z) = \epsilon$ and the proposition holds.

Let us assume that L is periodic and nontrivial. Let us first consider the case where $z = w \in \Delta^*$. Let $p = \mathit{Period}(L)$. We have:

$$\begin{aligned}\mathit{Shovel}(L, w) &= p^* \cdot \mathit{Prefix}(p)\\ \mathit{push}(L, w) &= p^i \cdot o \text{ for some } i > 0 \text{ and } o \in \mathit{Prefix}(p)\\ \mathit{offset}(L, w) &= (p^i \cdot o) \mod p = o\end{aligned}$$

Let us recall that $\mathit{push}(L, w)$ is also a prefix of w. Note that for any prefix u of a word v we have $u \cdot (u^{-1} \cdot v) = v$. Any word u in L is of the form p^k for some $k > 0$, and we have

$$\begin{aligned}\mathit{push}(L, w) \cdot (\mathit{offset}(L, w)^{-1} \cdot u \cdot \mathit{offset}(L, w)) \cdot \mathit{rest}(L, w)&\\ = p^i \cdot o \cdot (o^{-1} \cdot p^k \cdot o) \cdot (p^i \cdot o)^{-1} \cdot w&\\ = p^k \cdot p^i \cdot o \cdot (p^i \cdot o)^{-1} \cdot w&\\ = p^k \cdot w = u \cdot w&\end{aligned}$$

Let us now consider the case where $z = w^{-1}$ and $w \in \Delta^*$. Let $p^{\text{rev}} = \mathit{Period}(L^{\text{rev}})$. We have:

$$\begin{aligned}
\mathit{Shovel}(L^{\text{rev}}, w^{-1}) &= (p^{\text{rev}})^* \cdot \mathit{Prefix}(p^{\text{rev}}) \\
\mathit{push}(L^{\text{rev}}, w^{-1}) &= (p^{\text{rev}})^i \cdot o \text{ for some } i > 0 \text{ and } o \in \mathit{Prefix}(p^{\text{rev}}) \\
\mathit{offset}(L^{\text{rev}}, w^{-1}) &= ((p^{\text{rev}})^i \cdot o) \mod (p^{\text{rev}}) = o.
\end{aligned}$$

Let $u = p^k \in L$

$$\begin{aligned}
&\mathit{push}(L, w^{-1}) \cdot \big(\mathit{offset}(L, w^{-1})^{-1} \cdot u \cdot \mathit{offset}(L, w^{-1})\big) \cdot \mathit{rest}(L, w^{-1}) \\
&= (\mathit{push}(L^{\text{rev}}, w^{\text{rev}})^{\text{rev}})^{-1} \cdot \big(\mathit{offset}(L^{\text{rev}}, w^{\text{rev}})^{\text{rev}} \cdot u \cdot (\mathit{offset}(L^{\text{rev}}, w^{\text{rev}})^{\text{rev}})^{-1}\big) \\
&\quad \cdot (\mathit{rest}(L^{\text{rev}}, w^{\text{rev}})^{\text{rev}})^{-1} \\
&= (o^{\text{rev}} \cdot p^i)^{-1} \cdot \big(o^{\text{rev}} \cdot p^k \cdot (o^{\text{rev}})^{-1}\big) \cdot \left(\big(((p^{\text{rev}})^i \cdot o)^{-1} \cdot w^{\text{rev}})\big)^{\text{rev}}\right)^{-1}
\end{aligned}$$

Now, recall that for any u and v in Δ^*, $(u^{-1} \cdot v)^{\text{rev}} = v^{\text{rev}} \cdot (u^{\text{rev}})^{-1}$ and $(u \cdot v^{-1})^{-1} = v \cdot u^{-1}$. Thus we have

$$\begin{aligned}
\left(\big(((p^{\text{rev}})^i \cdot o)^{-1} \cdot w^{\text{rev}})\big)^{\text{rev}}\right)^{-1} &= \left((w^{\text{rev}})^{\text{rev}} \cdot \big(((p^{\text{rev}})^i \cdot o)^{\text{rev}}\big)^{-1}\right)^{-1} \\
&= \big((p^{\text{rev}})^i \cdot o\big)^{\text{rev}} \cdot w^{-1} \\
&= o^{\text{rev}} \cdot \big((p^{\text{rev}})^i\big)^{\text{rev}} \cdot w^{-1} \\
&= o^{\text{rev}} \cdot p^i \cdot w^{-1}
\end{aligned}$$

Hence,

$$\begin{aligned}
&\mathit{push}(L, w^{-1}) \cdot \big(\mathit{offset}(L, w^{-1})^{-1} \cdot u \cdot \mathit{offset}(L, w^{-1})\big) \cdot \mathit{rest}(L, w^{-1}) \\
&= (o^{\text{rev}} \cdot p^i)^{-1} \cdot \big(o^{\text{rev}} \cdot p^k \cdot (o^{\text{rev}})^{-1}\big) \cdot o^{\text{rev}} \cdot p^i \cdot w^{-1} \\
&= \big(p^{k-i} \cdot (o^{\text{rev}})^{-1}\big) \cdot o^{\text{rev}} \cdot p^i \cdot w^{-1}
\end{aligned}$$

Since, o^{rev} is a suffix of p, this gives $p^k \cdot w^{-1} = u \cdot w^{-1}$ and the lemma holds.

Consider L is non periodic and the case where $z = w \in \Delta^*$. Then there exists some word s such that $\mathit{Shovel}(L) = \mathit{Prefix}(s)$. Note that s is also $\mathit{Kernel}(L)$. In this case, $\mathit{push}(L, w) = \mathit{lcp}(\{w, s\})$ and therefore $\mathit{push}(L, w)$ is a prefix of s. We obtain that $\mathit{offset}(L, w) = \mathit{push}(L, w)$ and thus for all $u \in L$ we have $\mathit{push}(L, w) \cdot (\mathit{offset}(L, w)^{-1} \cdot u \cdot \mathit{offset}(L, w)) \cdot \mathit{rest}(L, w) = \mathit{push}(L, w) \cdot (\mathit{push}(L, w)^{-1} \cdot u \cdot \mathit{push}(L, w)) \cdot \mathit{push}(L, w)^{-1} \cdot w$. Recall that $\mathit{push}(L, w)$ is in $\mathit{Shovel}(L)$ and by the definition of *Shovel* we have that $\mathit{push}(L, w)$ is a prefix of $u \cdot \mathit{push}(L, w)$. Also, $\mathit{push}(L, w)$ is a prefix of w. Thus $\mathit{push}(L, w) \cdot (\mathit{push}(L, w)^{-1} \cdot u \cdot \mathit{push}(L, w)) \cdot \mathit{push}(L, w)^{-1} \cdot w = u \cdot w$.

The case where $z = w^{-1}$ and $w \in \Delta^*$ is similar. There exists some word $s = \mathit{Kernel}(L^{\text{rev}})$ such that $\mathit{Shovel}(L^{\text{rev}}) = \mathit{Prefix}(s)$, and $\mathit{push}(L^{\text{rev}}, w^{\text{rev}}) =$

$lcp(\{w,s\}) = \mathit{offset}(L^{\mathrm{rev}}, w^{\mathrm{rev}})$. Thus, for all $u \in L$ we have

$$
\begin{aligned}
&\mathit{push}(L, w^{-1}) \cdot (\mathit{offset}(L, w^{-1})^{-1} \cdot u \cdot \mathit{offset}(L, w^{-1})) \cdot \mathit{rest}(L, w^{-1})\\
&= (\mathit{push}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1} \cdot \big(\mathit{push}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}} \cdot u \cdot (\mathit{push}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}\big)\\
&\qquad \cdot \big((\mathit{push}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{-1} \cdot w^{\mathrm{rev}})^{\mathrm{rev}}\big)^{-1}\\
&= \big(u \cdot (\mathit{push}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}\big) \cdot \mathit{push}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}} \cdot w^{-1}\\
&= u \cdot w^{-1} \qquad \square
\end{aligned}
$$

C. Normalization

We prove here the Theorem 4.7, that we recall here:
Theorem 4.7 For an STW M let M' be the STW obtained with the method described in section 4.4. Then, M' is equivalent to M and satisfies $\mathbf{(E_1)}$ and $\mathbf{(E_2)}$.

This theorem is proven with the help of several auxiliary results.

Lemma C.1 *For any reduced language L we have*

1. *$\forall w \in \mathit{Prefix}(\mathit{Kernel}(L))$ the language $w^{-1} \cdot L \cdot w$ is reduced,*
2. *$\forall w \in \mathit{Suffix}(\mathit{Kernel}(L^{\mathrm{rev}}))$ the language $w \cdot L \cdot w^{-1}$ is reduced as well.*

PROOF We only prove 1 since 2 is a consequence of 1. If $\epsilon \in L$ then $\epsilon \in w^{-1} \cdot L \cdot w$ and the proposition is trivial. Otherwise, there exists two words u and v that differ on the first letter. But in this case $\mathit{Kernel}(L)$ is reduced to ϵ and the proposition is also trivially true. $\square$

Lemma C.2 *$\mathit{offset}(L, z)$ belongs to the set $\mathit{Prefix}(\mathit{Kernel}(L)) \cup \mathit{Suffix}(\mathit{Kernel}(L^{\mathrm{rev}}))$*

PROOF We prove the lemma for $z = w \in \Delta^*$, the other case being similar. We proceed by a case analysis depending on the class of L. If L is trivial then $\mathit{offset}(L, z) = \epsilon$. If L is periodic, then $\mathit{Shovel}(L) = \mathit{Kernel}(L)^* \cdot \mathit{Prefix}(\mathit{Kernel}(L))$. Therefore, $\mathit{push}(L, w)$ is of the form $\mathit{Kernel}(L)^i \cdot o$ where $i \geq 0$ and o is a prefix of $\mathit{Kernel}(L)$. Therefore $\mathit{offset}(L, w) = (\mathit{Kernel}(L)^i \cdot o) \bmod \mathit{Kernel}(L) = o$. Otherwise, L is not periodic and not trivial, $\mathit{push}(L, w) = lcp(\{w, \mathit{Kernel}(L)\})$ and therefore it is a prefix of $\mathit{Kernel}(L)$. By the definition of *offset* we have $\mathit{offset} = \mathit{push}(L, w)$ and the Lemma holds. $\square$

Lemma C.3 *For every $q \in Q$, every $z \in \mathit{Offsets}(q)$, and every $t \in \mathit{dom}(q)$*

$$[\![M]\!]_{\langle q,z\rangle}(t) = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot [\![M]\!]_q(t) \cdot \mathit{Right}(q)^{-1} \cdot z \tag{C.1}$$

PROOF Rules are built using the following algorithm.

```
1: $z_k := \mathit{Right}(p_k) \cdot u_k \cdot \mathit{Right}(p)^{-1} \cdot z$
2: **for** $i := k, \dots, 1$ **do**
3:     $u'_i := \mathit{rest}(p_i, z_i)$
4:     $p'_i := \langle p_i, \mathit{offset}(p_i, z_i)\rangle$
5:     $z_{i-1} := \mathit{Right}(p_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(p_i) \cdot \mathit{push}(p_i, z_i)$
6: $u'_0 := z^{-1} \cdot \mathit{Left}(p)^{-1} \cdot z_0$
```

We prove the lemma by induction on the structure of terms. For the base case, we consider constants and therefore rules of the form $\delta'(\langle q, z\rangle, a) = u$ and $T_q(a) = u$. The algorithm just realizes the affectation[1]:

$$u' = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot u \cdot \mathit{Right}(q)^{-1} \cdot z$$

Therefore $[\![M]\!]_{\langle q,z\rangle}(a) = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot u \cdot \mathit{Right}(q)^{-1} \cdot z$ and the lemma holds.

Let us now consider a term $t = f(t_1, \ldots, t_k)$, a word z and a rule $\delta(q, f) = u_0 \cdot q_1 \cdots q_k \cdot u_k$. Let us assume as our induction hypothesis that Equation (2) holds for each subterm in $t_1, \ldots, t_k$. We need to prove that:

$$\begin{aligned} &u'_0 \cdot [\![M]\!]_{q'_1}(t_1) \cdot u'_1 \cdots [\![M]\!]_{q'_k}(t_k) \cdot u'_k = \\ &\qquad z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot u_0 \cdot [\![M]\!]_{q_1}(t_1) \cdots [\![M]\!]_{q_k}(t_k) \cdot u_k \cdot \mathit{Right}(q)^{-1} \cdot z \end{aligned}$$

To apply our induction hypothesis, we need to verify the following $\mathit{offset}(q_i, z_i)$ belongs to the set $\mathit{Prefix}(\mathit{Kernel}(L_{q_i})) \cup \mathit{Suffix}(\mathit{Kernel}(L^{\mathrm{rev}}_{q_i}))$ and this is obtained by Lemma C.2.

Fact $\mathit{push}(q_i, z_i) \cdot [\![M]\!]_{q'_i}(t_i) \cdot \mathit{rest}(q_i, z_i) = \mathit{Left}(q_i)^{-1} \cdot [\![M]\!]_{q_i}(t_i) \cdot \mathit{Right}(q_i)^{-1} \cdot z_i$

Proof By induction hypothesis, we develop $[\![M]\!]_{q'_i}(t_i) = \mathit{offset}(q_i, z_i)^{-1} \cdot \mathit{Left}(q_i)^{-1} \cdot [\![M]\!]_{q_i}(t_i) \cdot \mathit{Right}(q_i)^{-1} \cdot \mathit{offset}(q_i, z_i)$. We observe that $\mathit{Left}(q_i)^{-1} \cdot [\![M]\!]_{q_i}(t_i) \cdot \mathit{Right}(q_i)^{-1}$ is a word of $L^{\circ}_{q_i}$, thus we conclude using Proposition 4.5. □

We prove the following invariant of the algorithm for every $1 \leq i \leq k$:

$$\begin{aligned} &z_{i-1} \cdot [\![M]\!]_{q'_i}(t_i) \cdot u'_i \cdots [\![M]\!]_{q'_k}(t_k) \cdot u'_k = \\ &\qquad \mathit{Right}(q_{i-1}) \cdot u_{i-1} \cdot [\![M]\!]_{q_i}(t_i) \cdot u_i \cdots [\![M]\!]_{q_k}(t_k) \cdot u_k \cdot \mathit{Right}(q)^{-1} \cdot z \end{aligned} \tag{C.2}$$

We proceed by induction on i from k to 1. For the base case $i = k$ we have:

$$\begin{aligned} &z_{k-1} \cdot [\![M]\!]_{q'_k}(t_k) \cdot u'_k \\ &\quad = \mathit{Right}(q_{k-1}) \cdot u_{k-1} \cdot \mathit{Left}(q_k) \cdot \mathit{push}(q_k, z_k) \cdot [\![M]\!]_{q'_k}(t_k) \cdot \mathit{rest}(q_k, z_k) \\ &\quad = \mathit{Right}(q_{k-1}) \cdot u_{k-1} \cdot \mathit{Left}(q_k) \cdot \\ &\qquad\quad \mathit{Left}(q_k)^{-1} \cdot [\![M]\!]_{q_k}(t_k) \cdot \mathit{Right}(q_k)^{-1} \cdot \mathit{Right}(q_k) \cdot u_k \cdot \mathit{Right}(q)^{-1} \cdot z \\ &\quad = \mathit{Right}(q_{k-1}) \cdot u_{k-1} \cdot [\![M]\!]_{q_k}(t_k) \cdot u_k \cdot \mathit{Right}(q)^{-1} \cdot z \end{aligned}$$

Thus the invariant (C.2) holds for the base case. Let us now consider it holds

[1]maybe make this case more clear in section Normalisation

for some $1 \leq i \leq k$. For $i-1$ we have

$$\begin{aligned}
&z_{i-1} \cdot [\![M]\!]_{q'_i}(t_i) \cdot u'_i \cdots [\![M]\!]_{q'_k}(t_k) \cdot u'_k \\
&\quad = \mathit{Right}(q_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(q_i) \cdot \mathit{push}(q_i, z_i) \cdot [\![M]\!]_{q'_i}(t_i) \cdot \mathit{rest}(q_i, z_i) \\
&\qquad \cdots [\![M]\!]_{q'_k}(t_k) \cdot u'_k \\
&\quad = \mathit{Right}(q_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(q_i) \cdot \mathit{Left}(q_i)^{-1} \cdot [\![M]\!]_{q_i}(t_i) \cdot \mathit{Right}(q_i)^{-1} \cdot z_i \\
&\qquad \cdots [\![M]\!]_{q'_k}(t_k) \cdot u'_k \\
&\quad = \mathit{Right}(q_{i-1}) \cdot u_{i-1} \cdot [\![M]\!]_{q_i}(t_i) \cdot \mathit{Right}(q_i)^{-1} \cdot \mathit{Right}(q_i) \cdot u_i \\
&\qquad \cdots [\![M]\!]_{q_k}(t_k) \cdot u_k \cdot \mathit{Right}(q)^{-1} \cdot z \\
&\quad = \mathit{Right}(q_{i-1}) \cdot u_{i-1} \cdot [\![M]\!]_{q_i}(t_i) \cdot u_i \cdots [\![M]\!]_{q_k}(t_k) \cdot u_k \cdot \mathit{Right}(q)^{-1} \cdot z
\end{aligned}$$

This proves the invariant. Now, since $u'_0 = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot z_0$, we obtain that

$$\begin{aligned}
&u'_0 \cdot [\![M]\!]_{q'_1} \cdot u'_1 \cdots [\![M]\!]_{q'_k} \cdot u'_k = \\
&\qquad z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot u_0 \cdot [\![M]\!]_{q_1}(t_1) \cdots [\![M]\!]_{q_k}(t_k) \cdot u_k \cdot \mathit{Right}(q)^{-1} \cdot z
\end{aligned}$$

□

Lemma C.4 **($\mathbf{E_1}$)** *is satisfied by M'.*

Proof To prove **($\mathbf{E_1}$)**, we need to prove that $L_{\langle q,z \rangle}$ is reduced. By previous lemma, we have for every $q \in Q$, every $z \in \mathit{Offsets}(q)$, and every $t \in \mathit{dom}(q)$

$$L_{\langle q,z \rangle} = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot L_q \cdot \mathit{Right}(q)^{-1} \cdot z$$

Since $z \in \mathit{Offsets}(q)$, then $z = w \in \mathit{Shovel}(L^\circ_q)$ or $z = w^{-1}$ with $w \in \mathit{Shovel}({L^\circ}_q^{\mathrm{rev}})$. The languages $\mathit{Left}(q)^{-1} \cdot L_q \cdot \mathit{Right}(q)^{-1} = {L^\circ}_q$ are reduced. Therefore using Proposition C.1, we obtain that **($\mathbf{E_1}$)** is satisfied. □

Lemma C.5 *Let L be a reduced language. For any $w \in \mathit{Prefix}(\mathit{Left}(L \cdot L'))$ we have $w \in \mathit{Prefix}(L')$, $\mathit{push}(L, w) = w$ and $\mathit{rest}(L, w) = \epsilon$.*

Proof If $w = \epsilon$ then the lemma is trivial. Otherwise, we know that $\epsilon \in L$ otherwise L would not be reduced. Hence, w is a common prefix of all words in L'. Therefore, $L \cdot w$ has also w as a common prefix and by definition $w \in \mathit{Shovel}(L)$. It follows that $\mathit{push}(L, w) = w$ and $\mathit{rest}(L, w) = \epsilon$. □

Lemma C.6 *For any rule $\delta'(\langle q, z \rangle, f) = u'_0 \cdot q'_1 \cdot u'_1 \cdots u'_{k-1} \cdot q'_k \cdot u'_k$ of M', for every $0 \leq i \leq k$ $u'_i \in \Delta^*$.*

Proof We consider a rule $\delta'(\langle q, z \rangle, f) = u'_0 \cdot q'_1 \cdot u'_1 \cdots u'_{k-1} \cdot q'_k \cdot u'_k$ of M' and $w \in \mathit{Suffix}(\mathit{Kernel}({L^\circ}_q^{\mathrm{rev}})) \cup \mathit{Prefix}(\mathit{Kernel}({L^\circ}_q))$. We essentially have to show that for all z_i computed by the normalisation algorithm, if $z_i = w_i^{-1}$ with $w_i \in \Delta^*$, then $u_i = \mathit{rest}({L^\circ}_{q_i}^{\mathrm{rev}}, w_i^{\mathrm{rev}}) = \epsilon$.

Let us consider L_i defined by:

$$L_i = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot u_0 \cdot \mathit{Left}(L_{q_1}) \cdot L^\circ_{q_1} \cdot \mathit{Right}(L_{q_1}) \cdot u_1 \cdots u_{i-1} \cdot \mathit{Left}(L_{q_i})$$

Observe that if w_i is a suffix of $\mathit{Right}(L_i \cdot L^\circ_{q_i})$ then w_i^{rev} is a prefix of $\mathit{Left}({L^\circ}^{\mathrm{rev}}_{q_i} \cdot L_i^{\mathrm{rev}})$. The language ${L^\circ}^{\mathrm{rev}}_{q_i}$ is reduced and by Lemma C.5, $\mathit{rest}(L^{\mathrm{rev}}_{q_i}, w_i^{\mathrm{rev}}) = \epsilon$ and $\mathit{push}(L^{\mathrm{rev}}_{q_i}, w_i^{\mathrm{rev}}) = w_i^{\mathrm{rev}}$. So it suffices to prove that for every i, w_i is a suffix of $\mathit{Right}(L_i \cdot L^\circ_{q_i})$.

We proceed by induction on i from k to 1. For the base case, $z_k = w_k^{-1} = \mathit{Right}(q_k) \cdot u_k \cdot \mathit{Right}(q)^{-1} \cdot z$. We have $L_{\langle q,z\rangle} = L_k \cdot {L^\circ}_{q_k} \cdot w_k^{-1}$ and according to Lemma C.4, $L_{\langle q,z\rangle}$ is reduced. Therefore $w_k = \mathit{Right}(L_k \cdot {L^\circ}_{q_k})$.

For the induction case, we have by induction hypothesis that w_i is a suffix of $\mathit{Right}(L_i \cdot {L^\circ}_{q_i})$, and from Lemma C.5, w_i is a suffix of $\mathit{Right}(L_i)$ and $\mathit{push}(L^{\mathrm{rev}}_{q_i}, w_i^{\mathrm{rev}}) = w_i^{\mathrm{rev}}$. From the algorithm $z_{i-1} = \mathit{Right}(q_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(q_i) \cdot \mathit{push}(q_i, z_i)$. Hence, $z_{i-1} = \mathit{Right}(q_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(q_i) \cdot w_i^{-1}$. If $z_{i-1} \in \Delta^*$ we are done. Otherwise $z_{i-1} = w_{i-1}^{-1}$ with $w_{i-1} \in \Delta^*$. Since w_i is a suffix of $\mathit{Right}(L_i) = \mathit{Right}(L_{i-1} \cdot \mathit{Right}(q_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(q_i))$, then z_{i-1} is a suffix of $\mathit{Right}(L_{i-1} \cdot {L^\circ}_{q_{i-1}})$. □

Lemma C.7 *Given a reduced language $L \subseteq \Delta^*$ and a word $z \in \mathbb{G}_\Delta$*

$$\mathit{lcp}((\mathit{offset}(L, z)^{-1} \cdot L \cdot \mathit{offset}(L, z)) \cdot \mathit{rest}(L, z)) = \epsilon$$

Proof Let us consider the case where $z = w \in \Delta^*$, the other case being symmetric. Let us remark using Lemma C.1, that $L' = \mathit{offset}(L, z)^{-1} \cdot L \cdot \mathit{offset}(L, z)$ is reduced. If L' does not contain ϵ then the Lemma is trivial. So let us consider the case where $\epsilon \in L'$.

We prove the lemma by contradiction and consider $u \neq \epsilon$ such that $u = \mathit{lcp}((\mathit{offset}(L, w)^{-1} \cdot L \cdot \mathit{offset}(L, w)) \cdot \mathit{rest}(L, w))$. Let $v = \mathit{push}(L, w) \cdot u$. We prove that v is a prefix of w. Indeed, since $\epsilon \in L'$, u is a prefix of $\mathit{rest}(L, w) = \mathit{push}(L, w)^{-1} \cdot w$. Using Proposition 4.5, we have that v is a common prefix of $L \cdot w$. Therefore, v belongs to $\mathit{Shovel}(L)$. This contradicts the fact that is the maximum prefix of w that belongs to $\mathit{Shovel}(L)$. □

Lemma C.8 **$(\mathbf{E_2})$** *is satisfied by M'.*

Proof According to Lemma C.4, each $L_{\langle q,z\rangle}$ is reduced. For the init rule, $(\mathbf{E_2})$ is direct consequence of Lemma C.7.

Consider a rule $\delta'(f, q') = u'_0 \cdot q_1 \cdots q_k \cdot u'_k$. We know from Lemma C.6 that each u'_i is either ϵ or a word in Δ^*. Moreover, according to line 3 of the algorithm, for for each $i > 0$, $u'_i = \mathit{rest}(q_i, z_i)$. Therefore, by Lemma C.7, $\mathit{lcp}(L_{q'_i} \cdot u'_i) = \epsilon$. Note that if two languages L and L' are such that $\mathit{lcp}(L) = \mathit{lcp}(L') = \epsilon$ then $\mathit{lcp}(L \cdot L') = \epsilon$. So, $\mathit{lcp}(L_{q'_i} \cdot u'_i \cdots L_{q'_k} \cdot u'_k) = \epsilon$ and $(\mathbf{E_2})$ is thus satisfied for every rule. □

Lemma C.9 *M is equivalent to M'*

Proof We prove that for all t, $u_0 \cdot [\![M]\!]_{q_0}(t) \cdot u_1 = u_0' \cdot [\![M]\!]_{q_0'}(t) \cdot u_1'$. To do that, we first inspect the initial rule. Using Lemma C.3, we have, with $v = \mathit{Right}(q_0) \cdot u_1$:

$$\begin{aligned}
u_0' \cdot [\![M]\!]_{q_0'}(t) \cdot u_1' &= u_0 \cdot \mathit{Left}(q_0) \cdot \mathit{push}(q_0, v) \\
&\quad \cdot \mathit{offset}(q_0, v)^{-1} \cdot \mathit{Left}(q_0)^{-1} \cdot [\![M]\!]_{q_0}(t) \cdot \mathit{Right}(q_0)^{-1} \cdot \mathit{offset}(q_0, v) \\
&\quad \cdot \mathit{rest}(q_0, v) \\
&= u_0 \cdot \mathit{Left}(q_0) \cdot \mathit{Left}(q_0)^{-1} \cdot [\![M]\!]_{q_0}(t) \cdot \mathit{Right}(q_0)^{-1} \cdot v \\
&\qquad \text{(using Prop. 4.5)} \\
&= u_0 \cdot [\![M]\!]_{q_0}(t) \cdot u_1
\end{aligned}$$

□

D. Myhill-Nerode Theorem

We present here omitted proof of section 5

Lemma 5.5 For any transformation τ and any $f \in \mathit{npaths}(\mathit{dom}(\tau))$, τ has at most one decomposition for f.

Proof Take any two decompositions $(u_0, \tau_1, u_1, \dots, \tau_k, u_k)$ and $(u_0', \tau_1', u_1', \dots, \tau_k', u_k')$ of τ for f. We first observe that $\mathit{dom}(\tau_i) = \mathit{dom}(\tau_i')$ for $1 \le i \le n$ by $\mathbf{(D_1)}$. Next, we show with an inductive argument that the corresponding components of the two decomposition are equal.

Assume then, that for some $0 \le i \le k$ two conditions hold (IH): 1) $u_j = u_j'$ for $0 \le j < i$ and 2) $\tau_j = \tau_j'$ for $0 < j < i$. We first show that $u_i = u_i'$. Observe that by $\mathbf{(D_2)}$ for $f(t_1, \dots, t_k) \in \mathit{dom}(\tau)$

$$u_0 \cdot \tau_1(t_1) \cdot \ldots \cdot \tau_k(t_k) \cdot u_k = u_0' \cdot \tau_1'(t_1) \cdot \ldots \cdot \tau_k'(t_k) \cdot u_k',$$

and by IH we get

$$u_i \cdot \tau_i(t_i) \cdot \ldots \cdot \tau_k(t_k) \cdot u_k = u_i' \cdot \tau_i'(t_i) \cdot \ldots \cdot \tau_k'(t_k) \cdot u_k'.$$

Therefore u_i is a prefix of u_i' or vice versa. W.l.o.g. assume that u_i is a prefix of u_i' i.e., $u_i' = u_i \cdot v$, and hence for any $f(t_1, \dots, t_k) \in \mathit{dom}(\tau)$

$$u_i' \cdot v \cdot \tau_i(t_i) \cdot \ldots \cdot \tau_k(t_k) \cdot u_k = u_i' \cdot \tau_i'(t_i) \cdot \ldots \cdot \tau_k'(t_k) \cdot u_k'.$$

Consequently, v is a prefix common to $\mathit{ran}(\tau_i) \cdot u_i \dots \mathit{ran}(\tau_k) \cdot u_k$ which by $\mathbf{(C_2)}$ may only be ε, and hence, $u_i = u_i'$. This also shows that for any $f(t_1, \dots, t_k) \in \mathit{dom}(\tau)$ we have

$$\tau_i(t_i) \cdot \ldots \cdot \tau_k(t_k) \cdot u_k = \tau_i'(t_i) \cdot \ldots \cdot \tau_k'(t_k) \cdot u_k'. \qquad \text{(D.1)}$$

Now, we fix some $f(s_1, \dots, s_k) \in \mathit{dom}(\tau)$ and assign

$$w = u_i \cdot \tau_{i+1}(s_{i+1}) \cdot \ldots \cdot \tau_k(s_k) \cdot u_k \quad w' = u_i' \cdot \tau_{i+1}'(s_{i+1}) \cdot \ldots \cdot \tau_k'(s_k) \cdot u_k'.$$

By (D.1) w is a suffix of w' or vice versa. W.l.o.g. we assume that w is a suffix of w' i.e., $w' = v \cdot w$. Now, for every $t_i \in \mathit{dom}(\tau_i) = \mathit{dom}(\tau_i')$ we have

$$\tau_i(t_i) \cdot w = \tau_i'(t_i') \cdot w', \quad \text{and thus,} \quad \tau_i(t_i) = \tau_i'(t_i) \cdot v.$$

By $\mathbf{(C_1)}$ τ_i is reduced, and in particular, $\mathit{lcs}(\mathit{ran}(\tau_i)) = \varepsilon$. Note that v is a suffix common to all images of τ_i must be ε, and therefore, $\tau_i = \tau_i'$. □

Lemma 5.13 $M/_{\equiv_M}$ is the minimal unique eSTW defining $[\![M]\!]$.

Proof Let $M/_{\equiv_M} = (\Sigma, \Delta, Q', \mathit{init}', \delta')$ and $\mathit{init}' = u_0' \cdot q_0' \cdot u_1'$. We take any eSTW $M'' = (\Sigma, \Delta, Q'', \mathit{init}'', \delta'')$ that is equivalent to M and let $\mathit{init}'' = u_0'' \cdot q_0'' \cdot u_1''$.

First, we show that M'' has at least as many states as $M/_{\equiv_M}$. For every $q' \in Q'$ we chose an arbitrary path $p_{q'}$ such that $\delta'(q_0', p_{q'}) = q'$ (there is at least one such path because we work with trimmed transducers) and we define $\lambda(q') = \delta''(q_0'', p_{q'})$, essentially the corresponding state of M''. We claim that λ is injective i.e., $\lambda(q_1') = \lambda(q_2')$ implies $q_1' = q_2'$. Let $q_1'' = \lambda(q_1')$ and $q_2'' = \lambda(q_2')$. Note that $[\![M'']\!]_{q_1''} = [\![M'']\!]_{q_2''}$, which by Lemma 5.12 gives us $[\![M']\!]_{q_1'} = [\![M']\!]_{q_2'}$. Recall that q_1' and q_2' are collections of states of M and from the construction of $M/_{\equiv_M}$ we have that $[\![M]\!]_{q_1} = [\![M]\!]_{q_2}$ for any $q_1 \in q_1'$ and any $q_2 \in q_2'$. Consequently, $q_1 \equiv_M q_2$ for any $q_1 \in q_1'$ and any $q_2 \in q_2'$, which shows that $q_1' = q_2'$.

Similarly, we use Lemma 5.12 to show that for every rule of $M/_{\equiv_M}$ the transducer M'' contains its exact copy (modulo state renaming λ). Therefore, either M'' contains more states or rules than $M/_{\equiv_M}$ M'' or M'' has the same number of states, the same number of rules that are identical to those of $M/_{\equiv_M}$ (modulo state renaming λ). □

Lemma 5.12 Take two eSTWs $M = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ and $M' = (\Sigma, \Delta, Q', \mathit{init}', \delta')$ defining the same transformation $\tau = [\![M]\!] = [\![M']\!]$ and let $\mathit{init} = u_0 \cdot q_0 \cdot u_1$ and $\mathit{init}' = u_0' \cdot q_0' \cdot u_1'$. Then, $u_0 = u_0'$ and $u_1 = u_1'$, and for every $p \in \mathit{paths}(\mathit{dom}(T))$, we let $q = \delta(q_0, p)$ and $q' = \delta'(q_0', p)$, and we have

1. $[\![M]\!]_q = [\![M]\!]_{q'}$,
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 $0 \le i \le k$.

Proof First, note that for every $t \in \mathit{dom}(\tau)$ we have that

$$[\![M]\!](t) = u_0 \cdot [\![M]\!]_{q_0}(t) \cdot u_1 = u_0' \cdot [\![M']\!]_{q_0'}(t) \cdot u_1' = [\![M']\!](t).$$

Therefore, u_0 is a prefix of u_0' or u_0' is a prefix of u_0. W.l.o.g. we assume that u_0 is a prefix of u_0' i.e., $u_0' = u_0 \cdot v$. Thus, we obtain $u_0 \cdot [\![M]\!]_{q_0}(t) \cdot u_1 = u_0 \cdot v \cdot [\![M']\!]_{q_0'}(t) \cdot u_1'$, and $[\![M]\!]_{q_0}(t) \cdot u_1 = v \cdot [\![M']\!]_{q_0'}(t) \cdot u_1'$. Consequently, v is also a prefix of $L_{q_0} \cdot u_1$ which by $\mathbf{(E_2)}$ implies that $v = \varepsilon$.

Also, we note that u_1 is a suffix of u_1' or u_1' is a suffix of u_1'. Again, w.l.o.g. we assume that u_1 is a suffix of u_1' i.e., $u_1' = v \cdot u_1$. We get that $u_0 \cdot [\![M]\!]_{q_0}(t) \cdot u_1 = u_0 \cdot [\![M']\!]_{q_0'}(t) \cdot v \cdot u_1$ and $[\![M]\!]_{q_0}(t) = [\![M']\!]_{q_0'}(t) \cdot v$. This implies that v is a suffix of L_{q_0} but by $\mathbf{(E_1)}$ we get that v must be an empty word, and so $u_1 = u'_1$ and $[\![M]\!]_{q_0} = [\![M']\!]_{q'_0}$.

Next, we prove the following inductive argument. If for a path $p \in \mathit{paths}(\mathit{dom}(\tau))$, $q = \delta(q_0, p)$, and $q' = \delta'(q'_0, p)$ we have $[\![M]\!]_q = [\![M']\!]_{q'}$, then

1. $\delta(q, f)$ is defined if and only if $\delta'(q', f)$ is, for every $f \in \Sigma$, and
2. 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 $0 \leq i \leq k$ and $[\![M]\!]_{q_i} = [\![M']\!]_{q'_i}$ for every $1 \leq i \leq k$.

The first condition follows trivially from equality of the domains of $[\![M]\!]_q$ and $[\![M']\!]_{q'}$. We prove the second condition with induction over i: we assume that $u_j = u'_j$ and $[\![M]\!]_{q_j} = [\![M']\!]_{q'_j}$ for every $0 \leq j < i \leq k$ and show that $u_i = u'_i$ and then $[\![M]\!]_{q_{i+1}} = [\![M]\!]_{q'_{i+1}}$. The arguments we use are analogous to those used to show equalities for the initial rules.

Since $[\![M]\!]_q = [\![M']\!]_{q'}$, for every $t = f(t_1, \ldots, t_k) \in \mathit{dom}([\![M]\!]_q)$

$$\begin{aligned}[\![M]\!]_q(t) &= u_0 \cdot [\![M]\!]_{q_1}(t_1) \cdot u_1 \cdot \ldots \cdot [\![M]\!]_{q_k}(t_k) \cdot u_k \\ &= u'_0 \cdot [\![M']\!]_{q'_1}(t_1) \cdot u'_1 \cdot \ldots \cdot [\![M']\!]_{q'_k}(t_k) \cdot u'_k \qquad = [\![M']\!]_{q'}(t),\end{aligned}$$

which by IH give us the following equality

$$\begin{aligned}&u_i \cdot [\![M]\!]_{q_{i+1}}(t_{i+1}) \cdot u_{i+1} \cdot \ldots \cdot [\![M]\!]_{q_k}(t_k) \cdot u_k \\ =&u'_i \cdot [\![M']\!]_{q'_{i+1}}(t_{i+1}) \cdot u'_{i+1} \cdot \ldots \cdot [\![M']\!]_{q'_k}(t_k) \cdot u'_k.\end{aligned}$$

Because of this equality we have that u_i is a prefix of u'_i or u'_i is a prefix of u_i. W.l.o.g. we assume that u_i is a prefix of u'_i i.e., $u'_i = u_i \cdot v$. Then, v is also a prefix of $L_{q_{i+1}} \cdot u_{i+1} \cdot \ldots \cdot L_{q_k} \cdot u_k$ and by $\mathbf{(E_2)}$ $v = \varepsilon$. Thus, $u_i = u'_i$.

Now, we fix $t = f(t_1, \ldots, t_k) \in \mathit{dom}([\![M]\!]_q)$ and let $w = u_{i+1} \cdot [\![M]\!]_{q_{i+2}}(t_{i+2}) \cdot \ldots \cdot [\![M]\!]_{q_k}(t_k) \cdot u_k$ and $w' = u'_{i+1} \cdot [\![M']\!]_{q'_{i+2}}(t_{i+2}) \cdot \ldots \cdot [\![M']\!]_{q'_k}(t_k) \cdot u'_k$. Note that w is a suffix of w' or w' is a suffix of w. W.l.o.g. we assume that w is a suffix of w' i.e., $w' = v \cdot w$. We observe that for every $t'_{i+1} \in (f, i+1)^{-1}\mathit{dom}([\![M]\!]_q) = (f, i+1)^{-1}\mathit{dom}([\![M']\!]_{q'})$ we have

$$[\![M]\!]_{q_{i+1}}(t'_{i+1}) \cdot w = [\![M']\!]_{q'_{i+1}}(t'_{i+1}) \cdot w'$$

and then

$$[\![M]\!]_{q_{i+1}}(t'_{i+1}) = [\![M']\!]_{q'_{i+1}}(t'_{i+1}) \cdot v,$$

which implies that v is a suffix of $L_{q_{i+1}}$ (since $(f, i+1)^{-1}\mathit{dom}([\![M]\!]_q) = \mathit{dom}([\![M]\!]_{q_{i+1}})$). By $\mathbf{(E_1)}$ we have, however, that $v = \varepsilon$. Thus, $[\![M]\!]_{q_{i+1}} = [\![M']\!]_{q'_{i+1}}$. □

Lemma 5.15 For any eSTW M, $\mathit{Can}([\![M]\!]) = M/_{\equiv_M}$.

Proof Let $M/_{\equiv_M} = (\Sigma, \Delta, Q', \mathit{init}', \delta')$ and $\mathit{init}' = u'_0 \cdot q'_0 \cdot u'_1$. First, we note that by Lemma 5.9 for any $p_1, p_2 \in \mathit{paths}(\mathit{dom}([\![M]\!]))$ we have that $p_1 \equiv_{[\![M]\!]} p_2$ if and only if $\delta(q'_0, p_1) = \delta(q'_0, p_2)$. Next, we define a mapping λ of the states of $\mathit{Can}([\![M]\!])$ to the states $M/_{\equiv_M}$ as $\lambda([p]_{[\![M]\!]}) = \delta(q'_0, p)$. Clearly, λ is properly defined and it is easy to see that λ is fact a bijection (a state renaming). By Lemma 5.12 we get that both transducers are identical (modulo the state renaming λ). □

Theorem 5.17 Minimization of STWs i.e., deciding whether for a given STW and $K \geq 0$ there exists an equivalent STW of size at most K, is NP-complete.

Proof We show NP-completeness of $\texttt{MINIMIZE}_{\text{STW}}$ by reduction from the following variant of satisfiability, which is known to be NP-complete [29].

> **Problem**: $\texttt{SAT}_{\texttt{ONE-IN-THREE}}$
> **Input**: $\varphi \in \texttt{3CNF}$, i.e. a conjunction of clauses, where each clause is a disjunction of 3 literals (a literal is a variable or its negation).
> **Question**: Does there exists a valuation satisfying φ and such that in every clause of φ exactly one literal is true?

Take any $\texttt{3CNF}$ formula $\varphi = c_1 \wedge \ldots \wedge c_k$ over the set of Boolean variables $x_1, \ldots, x_n$, where $c_j = L_{j,1} \vee L_{j,2} \vee L_{j,3}$ is a disjunction of exactly 3 literals for $j \in \{1, \ldots, k\}$. We assume that $k \geq 1$ and that no two clauses are the same (modulo reordering). The constructed STW M_φ essentially takes on the input the clauses of the formula φ and outputs a single character $\mathbf{t}$. Additionally, the STW M_φ accepts (multiple copies) of the trivial clauses $x_i \vee \neg x_i$ and also produces the single character output $\mathbf{t}$. To prevent the retraction of the single character to the initial rule the transformation defined by the transducer also maps a simple constant d to the empty word.

To make the semantics of the constructed transducer clear we use input symbols that are very similar to the notations introduced above. In particular, the ternary symbols $c_1, \ldots, c_k$ are used to represent clauses together with the constants $x_1, \neg x_1, \ldots, x_n, \neg x_n$ (where $\neg x_i$ is a single constant). Each trivial clause $x_i \wedge \neg x_i$ is represented three times with the use of binary symbols $c_{i,1}$, $c_{i,2}$, and $c_{i,3}$. Formally, we use the input alphabet $\Sigma = \Sigma^{(3)} \cup \Sigma^{(2)} \cup \Sigma^{(0)}$, where

$$\Sigma^{(3)} = \{c_1, \ldots, c_k\}, \quad \Sigma^{(2)} = \{c_{1,1}, c_{1,2}, c_{1,3}, \ldots, c_{n,1}, c_{n,2}, c_{n,3}\},$$

$$\Sigma^{(0)} = \{x_1, \neg x_1, \ldots, x_n, \neg x_n\} \cup \{d\}.$$

The output alphabet contains exactly one symbol $\Delta = \{\mathbf{t}\}$. First, we present an example of the transformation that the constructed transducer defined. For $\varphi_0 = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee \neg x_3)$ the transformation $\tau_{\varphi_0} = [\![M_{\varphi_0}]\!]$ is

$$\tau_{\varphi_0}(c_1(x_1, \neg x_2, x_3)) = \mathbf{t}, \quad \tau_{\varphi_0}(c_2(x_2, x_4, \neg x_3)) = \mathbf{t},$$
$$\tau_{\varphi_0}(d) = \varepsilon, \qquad \tau_{\varphi_0}(c_{i,\ell}(x_i, \neg x_i)) = \mathbf{t}, \qquad \text{for } i \in \{1,2,3,4\} \text{ and } \ell \in \{1,2,3\}.$$

The transducer $M_\varphi = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ is constructed as follows. The states of M_φ are

$$Q = \{q_0\} \cup \{q_{x_1}, \ldots, q_{x_n}\} \cup \{q_{\neg x_1}, \ldots, q_{\neg x_n}\}.$$

Its initial rule is $\mathit{init} = q_0$ and the transition function is defined as follows:

1. $\delta(q_0, c_j) = \mathbf{t} \cdot q_{L_{j,1}} \cdot q_{L_{j,2}} \cdot q_{L_{j,3}}$ for every $1 \leq j \leq k$ with $c_j = L_{j,1} \vee L_{j,2} \vee L_{j,3}$;
2. $\delta(q_0, c_{i,\ell}) = \mathbf{t} \cdot q_{p_i} \cdot q_{\neg p_i}$ for $1 \leq i \leq n$ and $1 \leq \ell \leq 3$;
3. $\delta(q_{x_i}, x_i) = \varepsilon$ and $\delta(q_{\neg x_i}, \neg x_i) = \varepsilon$ for $1 \leq i \leq n$;

4. $\delta(q_0, d) = \varepsilon$.

The transducer M_φ defines a transformation τ_φ satisfying exactly the following equations (and no other equation):

$$\begin{aligned}
&\tau_\varphi(c_j(L_{j,1}, L_{j,2}, L_{j,3})) = \mathbf{t}, && \text{for } j \in \{1, \dots, k\},\\
&\tau_\varphi(c_{i,\ell}(x_i, \neg x_i)) = \mathbf{t}, && \text{for } i \in \{1, \dots, n\} \text{ and } \ell \in \{1, 2, 3\},\\
&\tau_\varphi(d) = \varepsilon.
\end{aligned}$$

We observe that the size of M_φ is $16k + 5k + 3$. Essentially, the reduction relies on the use of the trivial clauses $x_i \vee \neg x_i$ with three separate symbols $c_{i,1}$, $c_{i,2}$, and $c_{i,3}$ which allows a more compact representation of the transformation τ_φ if there exists a satisfying valuation. The main claim is

$$\varphi \in \mathtt{SAT}_{\mathtt{ONE\text{-}IN\text{-}THREE}} \iff (M_\varphi, 14n + 4k + 3) \in \mathtt{MINIMIZE}_{\text{STW}}.$$

For the *only if* part we take the valuation $V : \{x_1, \dots, x_n\} \to \{\mathbf{true}, \mathbf{false}\}$ witnessing $\varphi \in \mathtt{SAT}_{\mathtt{ONE\text{-}IN\text{-}THREE}}$ and construct an STW M_V obtained essentially from M_φ by *pushing down* the symbols $\mathbf{t}$ to the subtrees that correspond to the literals satisfied by V. Formally, the transducer M_V differs from M_φ only on the transition function:

1. $\delta_V(q_0, c_j) = q_{L_{j,1}} \cdot q_{L_{j,2}} \cdot q_{L_{j,3}}$ for every $j \in \{1, \dots, k\}$ with $c_j = L_{j,1} \vee L_{j,2} \vee L_{j,3}$;
2. $\delta_V(q_0, c_{i,\ell}) = q_{x_i} \cdot q_{\neg x_i}$ for $1 \le i \le n$ and $1 \le \ell \le 3$;
3. $\delta_V(q_{x_i}, x_i) = \mathbf{t}$ and $\delta_V(q_{\neg x_i}, \neg x_i) = \varepsilon$ for $1 \le i \le n$ such that $V(x_i) = \mathbf{true}$;
4. $\delta_V(q_{x_i}, x_i) = \varepsilon$ and $\delta_V(q_{\neg x_i}, \neg x_i) = \mathbf{t}$ for $1 \le i \le n$ such that $V(x_i) = \mathbf{false}$;
5. $\delta_V(q_0, d) = \varepsilon$.

M_φ and M_V are equivalent because V is the witness of $\varphi \in \mathtt{SAT}_{\mathtt{ONE\text{-}IN\text{-}THREE}}$. Also, the size of M_V is exactly $14n + 4k + 3$.

For the *if* part, take the STW $M = (\Sigma, \Delta, Q_M, \mathit{init}_M, \delta_M)$ that is equivalent to M_φ and whose size is at most $14n + 4k + 2$. Let $\mathit{init}_M = w_0 \cdot q_0 \cdot w_1$, and observe that both w_0 and w_1 are ε because $[\![M_\varphi]\!](h) = \varepsilon$. We observe that M needs to have at least $1 + 2n$ states because it is the Myhill-Nerode index of the domain $\mathit{dom}([\![M_\varphi]\!])$. Consequently, M needs to have at least

1. 1 initial rule of size 1,
2. k transition rules of arity 3 for the symbols c_j ($j \in \{1, \dots, k\}$),
3. $3n$ transition rules of arity 2 for the symbols $c_{i,\ell}$ ($i \in \{1, \dots, n\}$ and $\ell \in \{1, 2, 3\}$),
4. n transition rules of arity 0 for the symbols x_i ($i \in \{1, \dots, n\}$),
5. n transition rules of arity 0 for the symbols $\neg x_i$ ($i \in \{1, \dots, n\}$), and
6. 1 transition rules of arity 0 for the symbol d.

By ignoring the sizes of possible output strings in the rules, the size of all rules is at least $13n + 4k + 3$. Thus, there are at most n *tokens* (with symbol) $\mathbf{t}$ to be distributed among the rules. We claim that the tokens are *pushed down* to states that handle the variables and the way the tokens are assigned to particular states yields a satisfying valuation.

We first show that indeed exactly n tokens must be distributed among the rules. For $i \in \{1, \dots, n\}$ and $\ell \in \{1, 2, 3\}$ we define $q_{x_i,\ell} = \delta_M(q_0, c_{i,\ell} \cdot 1)$ and $q_{\neg x_i,\ell} = \delta_M(q_0, c_{i,\ell} \cdot 2)$. Also, for every $i \in \{1, \dots, n\}$ let R_i be the set of rules $\delta_M(q_0, c_{i,\ell})$, $\delta_M(q_{x_i,\ell}, x_i)$, and $\delta_M(q_{\neg x_i,\ell}, \neg x_i)$ over $\ell \in \{1, 2, 3\}$. We observe that for every $i \in \{1, \dots, n\}$ the rules in R_i use at least one $\mathbf{t}$. Furthermore, for any $\ell_1, \ell_2 \in \{1, 2, 3\}$ and any $i_1, i_2 \in \{1, \dots, n\}$ such that $i_1 \neq i_2$ we have $q_{\ell_1,i_1} \neq q_{\ell_2,i_2}$ or otherwise the domain of M would be different from the domain of τ_φ. Therefore, for two different $i_1 \neq i_2$ the sets of rules R_{i_1} and R_{i_2} are disjoint, and consequently, at least n separate tokens $\mathbf{t}$ are used among the rules.

Since at least n tokens are used, the transducer M has exactly $2n + 1$ states including the initial state q_0, and in particular, $q_{x_i,\ell_1} = q_{x_i,\ell_2}$ and $q_{\neg x_i,\ell_1} = q_{\neg x_i,\ell_2}$ for any $i \in \{1, \dots, n\}$ and $\ell \in \{1, 2, 3\}$ (or, again, the domain of M and τ_φ would be different). Consequently, from now on we drop ℓ from the subscript and we write simply q_{x_i} and $q_{\neg x_i}$. Similarly, we show that $\delta_M(q_0, c_j \cdot \ell) = L_{i,\ell}$ for every $j \in \{1, \dots, k\}$ and every $\ell \in \{1, 2, 3\}$ with $c_j = L_{j,1} \vee L_{j,2} \vee L_{j,3}$.

We now show that the $\mathbf{t}$ tokens are attributed among the rules $\delta_M(q_{x_i}, x_i)$ and $\delta_M(q_{\neg x_i}, \neg x_i)$ only, and furthermore, we claim that for every $1 \leq i \leq n$ exactly one $\mathbf{t}$ token is attributed to either $\delta_M(q_{x_i}, x_i)$ or $\delta_M(q_{\neg x_i}, \neg x_i)$. To prove this claim we capture the set of those $i \in \{1, \dots, n\}$ such that neither $\delta_M(q_{x_i}, x_i)$ nor $\delta_M(q_{\neg x_i}, \neg x_i)$ use $\mathbf{t}$:

$$R = \{i \mid 1 \leq i \leq n,\ \delta_M(q_{x_i}, x_i) = \varepsilon,\ \delta_M(q_{\neg x_i}, \neg x_i) = \varepsilon\}.$$

We note that for any $i \notin R$ because $\tau_\varphi(c_{i,1}(x_i, \neg x_i)) = \mathbf{t}$, exactly one of the rules for $\delta_M(q_{x_i}, x_i)$ and $\delta_M(q_{\neg x_i}, \neg x_i)$ has one occurrence of $\mathbf{t}$. That consumes $n - |R|$ tokens. On the other hand, for every $i \in R$ and every $\ell \in \{1, 2, 3\}$ because $\tau_\varphi(c_{i,\ell}(x_i, \neg x_i)) = \mathbf{t}$, every rule $\delta_M(q_0, c_{i,\ell})$ uses one $\mathbf{t}$ token. Together, this totals to $(n - |R|) + 3|R| = n + 2|R|$ tokens used. Therefore, R must be empty or too many tokens would be used. This allows us to properly define the following valuation V_M:

$$V_M(x_i) = \begin{cases} \mathbf{true} & \text{if } \delta(q_{x_i}, x_i) = \mathbf{t}, \\ \mathbf{false} & \text{if } \delta(q_{\neg x_i}, \neg x_i) = \mathbf{t}. \end{cases}$$

Naturally, V_M satisfies φ because M is equivalent to M_φ.

The membership of $\texttt{MINIMIZE}_{\text{STW}}$ to NP follows from the fact that testing the equivalence of STWs is known to be in PTIME [31]. Hence, a nondeterministic Turing machine needs to guess an STW M' whose size is lower than $\min\{|M|, K\}$ and then test the equivalence of M and M'. □

E. Learning STWs

We give here the proof of Theorem 6.1, recalled below

Theorem 6.1 Checking if there exists an STW consistent with a given sample is NP-complete.

We prove this theorem in two phases: First, we show that the consistency problem is in NP, and then, we show that it is in fact NP-hard.

Lemma E.1 $\mathtt{CONS}_{\text{STW}}$ *is in* NP.

PROOF To show the membership of $\mathtt{CONS}_{\text{STW}}$ to NP we introduce the notion of alignment trees. An *alignment tree* is a finite tree over the ranked alphabet $\Gamma = \bigcup_{i=0}^{N} \Gamma^{(i)}$, where $\Gamma^{(i)} = \Sigma^{(i)} \times (\Delta^*)^{i+1}$ and N is the maximum arity of a symbol in Σ. An alignment tree α yields the corresponding example $Ex(\alpha) = (In(\alpha), Out(\alpha))$ with In and Out being defined as follows:

$$In(\langle f, u_0, \ldots, u_1\rangle(\alpha_1, \ldots, \alpha_k)) = f(In(\alpha_1), \ldots, In(\alpha_k))$$

$$Out(\langle f, u_0, \ldots, u_1\rangle(\alpha_1, \ldots, \alpha_k)) = u_0 \cdot Out(\alpha_1) \cdot u_1 \cdot Out(\alpha_2) \cdot u_2 \cdot \ldots \cdot u_{k-1} \cdot Out(\alpha_k) \cdot u_k.$$

Note that for a pair (t, w) there might exists an large number of alignment trees that yield the given pair. The size of all such alignment tree is, however, proportional to the sum of sizes of t and w.

Intuitively, an alignment tree is a trace of a run of an STW on the input tree, and not surprising, we shall view an STW as a DTA over Γ. A sufficient condition for translating successfully an arbitrary DTA $A = (\Gamma, Q, q_0, \delta)$ over Γ to an STW follows: $(\star)$ for every $f \in \Sigma^{(k)}$ and every $q \in Q$, δ is defined on $(q, \langle f, u_0, \ldots, u_k\rangle)$ for at most one tuple $(u_0, \ldots, u_k)$. This condition allows a relatively straightforward translation.

It is trivial to show that a sample S is consistent if and only if there exists an DTA A that satisfies $(\star)$ and such that for every pair $(t, w) \in S$ the automaton A recognizes at least one alignment tree yielding (t, w). Also, it is a folklore result that for a set of trees T there exists a DTA B capturing T i.e., $T \subseteq [\![B]\!]$, if and only if there exist a DTA B' capturing T whose size is polynomial in sum of sizes of trees of T.

Hence, we construct a nondeterministic Turing machine for testing $\mathtt{CONS}_{\text{STW}}$ that works as follows: (1) for every example in S it guesses an alignment tree yielding the example, (2) guesses the definition of an DTA over Γ, and (3) verifies that the DTA accepts all alignment trees constructed in step (1). □

Lemma E.2 $\mathtt{CONS}_{\text{STWS}}$ *is* NP*-hard.*

PROOF To show NP-hardness we reduce the variant $\mathtt{SAT}_{\mathtt{ONE\text{-}IN\text{-}THREE}}$ of satisfiabily known to be NP-complete [29]. Recall from the proof of Theorem 4 [19] that a `3CNF` formula φ belongs to $\mathtt{SAT}_{\mathtt{ONE\text{-}IN\text{-}THREE}}$ if there exists a valuation that satisfies exactly one literal in every clause of φ.

Now, we fix a `3CNF` formula φ over the set of Boolean variables $x_1, \ldots, x_n$, where every clause $c_j = L_{j,1} \vee L_{j,2} \vee L_{j,3}$ is a disjunction of exactly 3 literals $(j \in \{1, \ldots, k\})$. The constructed sample consists of examples of a transformation that takes a clause of φ and returns a single character $\mathbf{t}$.

To make the representation of the clauses clear we shall use input symbols that are similar to the notations above: we use an n-ary symbol c to represent a clause, binary symbols $x_1, \ldots, x_n$ for variables, and two constants $\bullet$ and $\circ$. Furthermore, $x_i(\bullet, \circ)$ corresponds to the positive literal x_i and $x_i(\circ, \bullet)$ corresponds to the negative literal $\neg x_i$ while $x_i(\circ, \circ)$ means that the variable x_i is not used and $x_i(\bullet, \bullet)$ corresponds to both the positive and the negative literals being used. For instance, for the formula $\varphi_0 = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee \neg x_3)$ we construct the sample S_{φ_0} containing the following examples:

$$
\begin{array}{ll}
(c(x_1(\bullet,\circ), x_2(\circ,\bullet), x_3(\bullet,\circ), x_4(\circ,\circ)), \mathbf{t}), & (c(x_1(\circ,\circ), x_2(\bullet,\circ), x_3(\circ,\bullet), x_4(\bullet,\circ)), \mathbf{t}), \\
(c(x_1(\bullet,\bullet), x_2(\circ,\circ), x_3(\circ,\circ), x_4(\circ,\circ)), \mathbf{t}), & (c(x_1(\circ,\circ), x_2(\bullet,\bullet), x_3(\circ,\circ), x_4(\circ,\circ)), \mathbf{t}), \\
(c(x_1(\circ,\circ), x_2(\circ,\circ), x_3(\bullet,\bullet), x_4(\circ,\circ)), \mathbf{t}), & (c(x_1(\circ,\circ), x_2(\circ,\circ), x_3(\circ,\circ), x_4(\bullet,\bullet)), \mathbf{t}), \\
(c(x_1(\circ,\circ), x_2(\circ,\circ), x_3(\circ,\circ), x_4(\circ,\circ)), \varepsilon). &
\end{array}
$$

Essentially, the first two examples encode the two clauses of φ_0. The next four examples encode the trivial clauses $x_i \vee \neg x_i$, which ensure that every variable has a proper valuation. The last example encode the empty false clause, which ensures that the output $\mathbf{t}$ can originate from used literals only.

Formally, the input alphabet is $\Sigma = \{c^{(3)}, x_1^{(2)}, \ldots, x_n^{(2)}, \bullet^{(0)}, \circ^{(0)}\}$ and the output alphabet is $\Delta = \{\mathbf{t}\}$. The sample S_φ contains the following examples:

(1) $(c(x_1(\circ,\circ), \ldots, x_n(\circ,\circ)), \varepsilon)$

(2) $(c(x_1(\circ,\circ), \ldots x_{i-1}(\circ,\circ), x_i(\bullet,\bullet), x_{i+1}(\circ,\circ), \ldots, x_n(\circ,\circ)), \mathbf{t})$ for $i \in \{1, \ldots, n\}$,

(3) $(c(x_1(\ell^+_{j,1}, \ell^-_{j,1}), \ldots, x_n(\ell^+_{j,n}, \ell^-_{j,n})), \mathbf{t})$ for $j \in \{1, \ldots, m\}$, where $\ell^+_{j,i}$ is $\bullet$ if the clause c_j uses x_i and $\circ$ otherwise, and analogously, $\ell^-_{j,i}$ is $\bullet$ if the clause c_j uses $\neg x_i$ and $\circ$ otherwise (with $i \in \{1, \ldots, n\}$).

The main claim is

$$S_\varphi \in \texttt{CONS}_{\text{STW}} \iff \varphi \in \texttt{SAT}_{\texttt{ONE-IN-THREE}}.$$

For the *only if* part we take a valuation $V : \{x_1, \ldots, x_n\} \to \{\mathbf{true}, \mathbf{false}\}$ that witnesses $\varphi \in \texttt{SAT}_{\texttt{ONE-IN-THREE}}$. We define a STW $M_V = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ with the states $Q = \{q_i, q_{\langle i,+\rangle}, q_{\langle i,-\rangle} \mid i \in \{1, \ldots, n\}\} \cup \{q_0\}$, the initial rule $\mathit{init} = q_0$ and the following transitions (with $i \in \{1, \ldots, n\}$):

$$
\begin{array}{ll}
\delta(q_0, c) = q_1 \cdot \cdots \cdot q_n, & \delta(q_i, x_i) = q_{\langle i,+\rangle} \cdot q_{\langle i,-\rangle}, \\
\delta(q_{\langle i,+\rangle}, \circ) = \varepsilon, & \delta(q_{\langle i,-\rangle}, \circ) = \varepsilon, \\
\delta(q_{\langle i,+\rangle}, \bullet) = \begin{cases} \mathbf{t} & \text{if } V(x_i) = \mathbf{true}, \\ \varepsilon & \text{if } V(x_i) = \mathbf{false}, \end{cases} & \delta(q_{\langle i,-\rangle}, \bullet) = \begin{cases} \varepsilon & \text{if } V(x_i) = \mathbf{true}, \\ \mathbf{t} & \text{if } V(x_i) = \mathbf{false}. \end{cases}
\end{array}
$$

It is easy to verify that M_V is consistent with S_φ: consistency with examples (1) and (2) follows from the definition of M_V and consistency with examples (3) follows from V witnessing $\varphi \in \texttt{SAT}_{\texttt{ONE-IN-THREE}}$.

For the *if* part, we take an STW M consistent with S_φ and define the following collection of trees (with $i \in \{1, \dots, n\}$):

$$\begin{aligned} t_i^+ &= c(x_1(\circ,\circ), \dots, x_{i-1}(\circ,\circ), x_i(\bullet,\circ), x_{i+1}(\circ,\circ), \dots, x_n(\circ,\circ)), \\ t_i^- &= c(x_1(\circ,\circ), \dots, x_{i-1}(\circ,\circ), x_i(\circ,\bullet), x_{i+1}(\circ,\circ), \dots, x_n(\circ,\circ)). \end{aligned}$$

Essentially, t_i^+ corresponds to the clause x_i and t_i^- corresponds to the clause $\neg x_i$ only. Since the domain of M is path-closed and contains the trees from S_φ, M also accepts the trees t_i^+ and t_i^-. Because of the examples (1) and (2) for every $i \in \{1, \dots, n\}$ we have that either $[\![M]\!](t_i^+) = \mathbf{t}$ and $[\![M]\!](t_i^-) = \varepsilon$ or $[\![M]\!](t_i^+) = \varepsilon$ and $[\![M]\!](t_i^-) = \mathbf{t}$. Therefore, the following function is a properly defined valuation of the variables:

$$V_M(x_i) = \begin{cases} \textbf{true} & \text{if } [\![M]\!](t_i^+) = \mathbf{t}, \\ \textbf{false} & \text{if } [\![M]\!](t_i^-) = \mathbf{t}. \end{cases}$$

Now, V_M witnesses $\varphi \in \mathtt{SAT}_{\mathtt{ONE\text{-}IN\text{-}THREE}}$ because of the examples (3). $\square$