On the decomposition of finite-valued streaming string transducers

Paul Gallot, Anca Muscholl, Gabriele Puppis, Sylvain Salvati

On the decomposition of finite-valued streaming string transducers

Paul Gallot[1], Anca Muscholl[2], Gabriele Puppis[2], and Sylvain Salvati[3]

1   **ENS Paris-Saclay**
    `pgallot@ens-paris-saclay.fr`
2   **Université de Bordeaux, LaBRI & CNRS**
    `{pgallot,anca,gabriele.puppis}@labri.fr`
3   **Université de Lille 1, CRIStAL & INRIA**
    `sylvain.salvati@univ-lille1.fr`

— Abstract

We prove the following decomposition theorem: every 1-register streaming string transducer that associates a uniformly bounded number of outputs with each input can be effectively decomposed as a finite union of functional 1-register streaming string transducers. This theorem relies on a combinatorial result by Kortelainen concerning word equations with iterated factors. Our result implies the decidability of the equivalence problem for the considered class of transducers. This can be seen as a first step towards proving a more general decomposition theorem for streaming string transducers with multiple registers.

1   Introduction

Data management systems heavily depend upon models of transformation of data that must be efficient both in terms of processing time and in terms of memory resources, and yet enjoy decidable static analysis methods (e.g. algorithms for checking equivalence).

Finite transducers, that is, finite automata extended with outputs, are simple devices that enable some effective, and even efficient reasoning on data transformations. These models come in different variants, depending on whether they use non-determinism or not, on whether the input is scanned only once from left to right (one-way transducers) or several times and in different directions (two-way transducers, or two-way generalised sequential machines 2GSM), and on the number of outputs that can be associated with each input. Concerning the last property, transducers that associate at most one output with each input are called functional transducers.

One may expect that the transformations computed by finite state transducers can be equally described in logic, like monadic second-order logic (MSO), but this holds only up to a certain extent. For instance, two-way transducers are as expressive as MSO-definable transductions [6] when restricted to the functional case, but they become incomparable as soon as unrestricted (i.e. relational) transductions are considered.

Streaming String Transducers (SSTs) [2] have been introduced as a mechanical representation of transducers described by means of MSO. They gained a growing interest in a series of works [1, 2, 3, 4], where their formal properties have been investigated and where they have been put to applications to model transformations on data streams, heaps etc.

Intuitively, SSTs are one-way automata enhanced with a finite number of registers to store and manipulate partial outputs. The registers can be updated by appending and prepending symbols, and possibly by concatenating several registers into one; the register content, however, cannot be inspected to control the flow of the computation.

Contrary to two-way transducers, not only functional SSTs are as expressive as MSO-definable functional transductions, but also this remains true when considering relational transductions. Therefore, as for MSO transductions, functional SSTs are equally expressive as functional two-way transducers and incomparable to them for relational transductions. Somehow, the equivalence of SSTs with MSO transductions is rather intuitive in view of a result [8] that characterizes MSO tree transductions in terms of finite copying macro tree transducers with the single use restriction. In SSTs, the finite copying mechanism is represented by the use of a finite number of registers, while the single use restriction for SSTs is formalized as the requirement that, during a transition of the machine, the content of each register can be used at most once to produce the new contents (*copyless* SSTs). Moreover, this result relates SSTs to much older formalisms, notably the Lindenmayer's L systems called D0L and HDT0L, which can be used to describe transductions and whose generalizations to trees can be seen, in a certain way, as macro tree transducers.

This paper focuses on finite-valued transductions, which lie strictly between functional and relational ones. Formally, a transduction is finite-valued if it associates a uniformly bounded number of outputs (say, at most k outputs) with each input. For one-way transducers it is known how to decide whether it is finite-valued [17], and in this case compute the maximal number of outputs associated with some input [11]. While the decidability of this property is still an open question for two-way transducers. What is more interesting is that, even if finite-valued transducers capture more transductions than functional transducers, they often enjoy properties similar to the latter ones, especially concerning the decidability of the fundamental problems and the relationship with logic.

Of course, transducers beyond the classical one-way case pose new challenges with respect to the fundamental problems, notably, the equivalence problem. For example, equivalence is known to be decidable for functional SSTs, even without the copyless restriction [9], but it is undecidable for unrestricted (i.e. relational) ones [10]. On the other hand, the equivalence problem remains decidable for finite-valued 2GSMs [12], by an argument relying on Ehrenfeuchts's conjecture. The latter paper actually shows that every HDT0L language L has an effective test set, that is, a finite subset on which equivalence of finite-valued 2GSM w.r.t. L can be tested.

The pervasive similarity between finite-valued and functional transducers can sometimes be explained by a quite strong *decomposition theorem* of the following form, where $\mathscr{C}$ is an appropriate class of transducers:

Within $\mathscr{C}$ finite-valued transducers are equivalent to finite unions of functional transducers.

It is known from [18] that the above statement holds when $\mathscr{C}$ is the class of one-way transducers. Our main interest is in proving the decomposition theorem for the class of SSTs, since, as we will see, this implies solutions to a certain number of open problems, notably, the decidability of the equivalence problem for finite-valued SSTs and the equivalence to finite-valued two-way transducers. We begin this investigation by proving a decomposition theorem for the sub-class of SSTs with 1 register. Unfortunately, we are not able to generalize this result to SSTs with multiple registers. Towards the end of the paper we outline a few points of our proof that are problematic for the generalization.

2   Preliminaries

Two-way transducers. A *two-way transducer*, or *two-way generalized sequential machine*
(abbreviated *2GSM*), is a tuple $\mathcal{T} = (\Sigma, \Gamma, Q, I, E, F)$, where Σ (resp. Γ) is a finite input
(resp. output) alphabet, Q is a finite set of states, I (resp. F) is a set of initial (resp. final)
states included in Q, and $E \subseteq Q \times \Sigma \times \Gamma^\star \times Q \times \{-1, +1\}$ is a finite set of transition rules
describing, for each state and input symbol, the possible output string, target state, and
direction of movement. To correctly define the transitions at the extremities of the input,
we use two special symbols $\triangleright$ and $\triangleleft$ and assume that the input of a 2GSM is of the form
$w = a_1 \ldots a_n$, with $n \geq 2$, $a_1 = \triangleright$, $a_n = \triangleleft$, and $a_i \neq \triangleright, \triangleleft$ for all $i = 2, \ldots, n-1$. We say that $\mathcal{T}$
is *one-way* if it can only move to the right, that is, $(q, a, v, q', d) \in E$ implies $d = +1$.

A *configuration* of $\mathcal{T}$ on input w is a pair $(q, p) \in Q \times \{1, \ldots, |w|\}$ consisting of a state
and a position in the input. The *transitions* of $\mathcal{T}$ on w connect pairs of configurations and
are of the form $(q, p) \xrightarrow[\mathcal{T}]{a/v} (q', p')$, with $a = w(p)$, $(q, a, v, q', d) \in E$, and $p' = p + d$. We
assume that $\mathcal{T}$ can only move right (resp. left) at the first (resp. last) position of w, that is,
$(q, \triangleright, v, q', d) \in E$ implies $d = +1$ and $(q, \triangleleft, v, q', d) \in E$ implies $d = -1$. A *run* of $\mathcal{T}$ on w is any
sequence of transitions of the form $(q_0, p_0) \xrightarrow[\mathcal{T}]{w(p_0)/v_1} (q_1, p_1) \xrightarrow[\mathcal{T}]{w(p_1)/v_2} \ldots \xrightarrow[\mathcal{T}]{w(p_{n-1})/v_n} (q_n, p_n)$.
The *output* of a run ρ is the word $\mathsf{out}(\rho) = v_1 v_2 \ldots v_n$. The run ρ is *initial* (resp. *final*) if
$q_0 \in I$ and $p_0 = 1$ (resp. if $q_n \in F$ and $p_n = |w|$).

Streaming string transducers.   For convenience, we define streaming string trans-
ducers in a slightly different but equivalent way compared to [2] (this concerns in par-
ticular the definition of the output). A *streaming string transducer* (*SST*) is a tuple
$\mathcal{T} = (\Sigma, \Gamma, Q, X, x_{\mathsf{out}}, U, I, E, F)$, where Σ, Γ are the usual input and output alphabets, Q
is the state space, X is a finite set of registers disjoint from Γ, $x_{\mathsf{out}} \in X$ is a special register
used to describe the final output, U is a finite set of *register updates*, namely, functions from
X to $(X \uplus \Gamma)^*$, I is a subset of Q representing the initial states, $E \subseteq Q \times \Sigma \times U \times Q$ is a set
of transition rules, describing, for each state and input symbol, the possible updates and
target states, and $F \subseteq Q$ is a set of final states.

To define the semantics of $\mathcal{T}$, we make use of *valuations*, that is, functions $g : X \to \Gamma^*$. A
valuation g can be homomorphically extended to words over $X \uplus \Gamma$ and to register updates,
as follows. Given a word $w \in (X \uplus \Gamma)^*$, $g(w)$ is the word over Γ obtained from w by replacing
every occurrence of a register x with its valuation $g(x)$. Similarly, given an update $f : X \to (X \uplus \Gamma)^*$, $g \circ f$ is the valuation that maps each register x to the word $g(f(x))$. A *configuration*
of $\mathcal{T}$ is a pair state-valuation (q, g). The configuration is *initial* (resp. *final*) if $q \in I$ and
$g(x) = \varepsilon$ for all registers $x \in X$ (resp. $q \in F$). When reading a symbol a, the SST moves from
configuration (q, g) to configuration (q', g') if there is a transition rule $(q, a, f, q') \in E$ such
that $g' = g \circ f$. We denote such a transition by $(q, g) \xrightarrow[\mathcal{T}]{a/f} (q', g')$. A *run* of $\mathcal{T}$ on $w = a_1 \ldots a_n$
is any sequence of transitions of the form $(q_0, g_0) \xrightarrow[\mathcal{T}]{a_1/f_1} (q_1, g_1) \xrightarrow[\mathcal{T}]{a_2/f_2} \ldots \xrightarrow[\mathcal{T}]{a_n/f_n} (q_n, g_n)$. It is
initial (resp. *successful*) if it begins with an initial configuration (resp. if it begins with an
initial configuration and ends with a final configuration). The *output* of a successful run is
the last valuation $g_n(x_{\mathsf{out}})$ of the special register x_{out}.

A well-behaved class of SSTs is obtained by requiring that all the updates are copyless.
Formally, an SST $\mathcal{T} = (\Sigma, \Gamma, Q, X, x_{\mathsf{out}}, U, I, E, F)$ is *copyless* if for every update $f \in U$, every
register $x \in X$ appears at most once in $f(x_1) \cdot \ldots \cdot f(x_k)$, where $X = \{x_1, \ldots, x_k\}$. Hereafter
we tacitly assume that all SSTs are copyless.

Unambiguity, transductions, functionality.   The following definitions apply to both
two-way transducers and streaming string transducers. The *input automaton* of a transducer

$\mathcal{T}$ is a finite automaton $\mathcal{A}$ obtained from $\mathcal{T}$ by removing any information concerned with the construction of the output (e.g. the output alphabet, the registers, etc.). For example, if $\mathcal{T}$ is an SST, then the input automaton $\mathcal{A}$ of $\mathcal{T}$ has transition rules of the form (q, a, q') for all transition rules (q, a, f, q') of $\mathcal{T}$. This automaton recognizes precisely the language of input words w that have at least one successful run of $\mathcal{T}$ (and hence at least one associated output). We say that $\mathcal{T}$ is *input-unambiguous* (resp. *input-k-ambiguous*) if its input automaton is unambiguous, namely, it admits at most one successful run on each input (resp. if the input automaton admits at most k successful runs on each input).

A *transduction* is a relation $T \subseteq \Sigma^* \times \Gamma^*$ consisting of pairs of input and output words. It is *computed* by a transducer $\mathcal{T}$ if it contains exactly those pairs $(w, v) \in \Sigma^* \times \Gamma^*$ for which there is a successful run of $\mathcal{T}$ on w with output v. $\mathcal{T}$ is called *functional* if it computes a transduction which is a partial function, namely, it associates at most one output with each input. Similarly, it is *k-valued* if it associates at most k outputs with each input (in [4] such a transducer is called *finitary*).

Transductions can be also defined in monadic second-order logic (*MSO*). Intuitively, this is done by specifying an output (seen as a relational structure) from a fixed number of copies of the input. We only give a short account of the definitions and we refer the reader to [5] for the details. An *MSO transduction with m copies* consists of a formula $\Phi_{\mathsf{dom}}(\bar{Z})$, where $\bar{Z} = (Z_1, \ldots, Z_m)$ is an m-tuple of free monadic variables, a formula $\Phi_a^i(\bar{Z}, x)$ for each $i \in \{1, \ldots, m\}$ and $a \in \Gamma$, and a formula $\Phi_\le^{i,j}(\bar{Z}, x, y)$ for each $i, j \in \{1, \ldots, m\}$. For different valuations of the free variables $\bar{Z}$, which range over sets of positions of the input, we may have different outputs associated with that input. The formula $\Phi_{\mathsf{dom}}(\bar{Z})$ describes which valuations generate an output. The formula $\Phi_a^i(\bar{Z}, x)$ tells whether, in the output corresponding to the valuation of $\bar{Z}$, the position x in the i-th copy of the input is an element of the output and, in this case, if it is labeled with the letter a. Similarly, the formula $\Phi_\le^{i,j}(x, y)$ tells whether, in the output corresponding to the valuation of $\bar{Z}$, the element x of the i-th copy of the input precedes the element y of the j-th copy of the input.

3   The conjectured decomposition theorem and its implications

We conjecture the following decomposition theorem for SSTs:

▶ **Conjecture 1.** *k-valued SSTs are effectively equivalent to finite unions of functional SSTs.*

In Section 5 we will give a proof of this conjecture in the restricted case of SSTs with a single register. Even if we are not able to prove the result in its full generality, we discuss here some important implications of the conjecture, which motivate our interest in having such a result. A first consequence would be the decidability of equivalence for k-valued SSTs:

▶ **Corollary 2.** *Assuming that Conjecture 1 holds, one can decide if two k-valued SSTs $\mathcal{T}, \mathcal{T}'$ are equivalent.*

Proof. Conjecture 1 implies that $\mathcal{T} = \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$ and $\mathcal{T}' = \mathcal{T}'_1 \cup \cdots \cup \mathcal{T}'_k$ where $\mathcal{T}_1, \ldots, \mathcal{T}_k$, $\mathcal{T}'_1, \ldots, \mathcal{T}'_k$ are all functional transducers. From [4], we know that functional SSTs can be made deterministic, and also that the inclusion (as a transduction) of an SST within a finite union of deterministic SSTs is decidable. A consequence is that we can decide whether both $\mathcal{T} \subseteq \mathcal{T}'_1 \cup \ldots \cup \mathcal{T}'_k$ and $\mathcal{T}' \subseteq \mathcal{T}_1 \cup \ldots \cup \mathcal{T}_k$ hold, that is, whether $\mathcal{T}$ is equivalent to $\mathcal{T}'$.   ◀

Another implication concerns the relationship between SSTs and 2GSMs:

▶ **Corollary 3.** *Assuming that Conjecture 1 holds, k-valued SSTs and k-valued 2GSMs are equally expressive. In addition, this would give a decomposition theorem for k-valued 2GSMs.*

Proof. From [2] we know that functional SSTs are expressively equivalent to functional MSO-definable transductions, and from [7] the latter are expressively equivalent to functional 2GSMs. So functional SSTs can be translated to equivalent functional 2GSMs. Our conjecture would then imply that k-valued SSTs can be transformed to equivalent 2GSMs.

The proof that k-valued 2GSMs can be translated to equivalent SSTs is sketched below. Note that this translation does not rely on any decomposition theorem for 2GSMs. In fact, the previous arguments paired with the following proof, do imply a decomposition theorem for k-valued 2GSMs. Consider a 2GSM $\mathcal{T}$ and define the *crossing number* of a successful run ρ of $\mathcal{T}$ as the maximum number of configurations that occur in ρ at same position p of the underlying input. Using classical techniques from [16], one can encode with a regular language any set of successful runs of $\mathcal{T}$ with uniformly bounded crossing numbers. Using such an encoding and a number of registers proportional to the bound on the crossing numbers, one can then simulate the output produced by those runs in a single left-to-right pass, that is by means of an SST $\mathcal{T}'$. In general, the transduction computed by $\mathcal{T}'$ is included in that of $\mathcal{T}$. The crucial observation is that, if $\mathcal{T}$ is k-valued, then all its outputs can be produced by successful runs that have crossing number at most $|Q|$, where Q is the state space of $\mathcal{T}$. The SST $\mathcal{T}'$ is thus equivalent to $\mathcal{T}$.   ◂

4   The classical decomposition theorem for one-way transducers

We give a short overview of the proof of a classical decomposition theorem for one-way transducers, following the presentation of [15]:

▶ **Theorem 4** ([18])**.** *k-valued one-way transducers are effectively equivalent to finite unions of functional one-way transducers.*

We present this proof, not only for self-containment, but also because it allows us to isolate the key important properties that need to be generalized in order to obtain a decomposition theorem for SSTs. After the proof sketch, we will mention some important implications of the theorem, notably, the decidability of the equivalence problem and the correspondence with a suitable subclass of MSO-definable transductions.

Given two automata or transducers $\mathcal{A}, \mathcal{B}$, we say that $\mathcal{B}$ is a *covering* of $\mathcal{A}$ if there is a function h from the states of $\mathcal{B}$ to the states of $\mathcal{A}$ such that (i) h preserves the transitions, that is, if $\tau = (q, a, ..., q')$ is a transition rule of $\mathcal{B}$, then $h(\tau) =^{\text{def}} \big(h(q), a, ..., h(q')\big)$ is a transition rule of $\mathcal{A}$, (ii) h induces a bijection between the initial states of $\mathcal{B}$ and the initial states of $\mathcal{A}$, (iii) for all states q of $\mathcal{B}$, h induces a bijection between the transitions of $\mathcal{B}$ exiting q and the transitions of $\mathcal{A}$ exiting $h(q)$, and (iv) both h and h^{-1} preserve final states, that is, q is final in $\mathcal{A}$ iff $h(q)$ is final in $\mathcal{B}$. Note that when $\mathcal{B}$ is a covering of $\mathcal{A}$, there is a bijection between the (successful) runs of $\mathcal{B}$ and the (successful) runs of $\mathcal{A}$; in particular, $\mathcal{B}$ and $\mathcal{A}$ are equivalent (they recognize the same language or compute the same transduction). A common way to define a covering of $\mathcal{A}$ consists in equipping the states of $\mathcal{A}$ with additional information which can be maintained in a deterministic way during the transitions. In this case, the states of the covering can be denoted by pairs of the form (q, o), where q is a state of $\mathcal{A}$ and o is some object (the additional information) associated with q. Accordingly, given a run ρ of such a covering of $\mathcal{A}$, we denote by $\rho|_1$ the corresponding run of $\mathcal{A}$.

Let $\mathcal{T}$ be a one-way transducer and ρ, ρ' two runs of it on the same input and with possibly different outputs $\mathsf{out}(\rho), \mathsf{out}(\rho')$. We would like to give a "measure" of the divergence of these outputs. The size of a pair of words $(u, v) \in \Gamma^* \times \Gamma^*$ is the sum of the lengths of u and v, i.e. $|(u, v)| = |u| + |v|$. The *lead or delay* of a pair of words (x, y), denoted $\mathsf{LD}(x, y)$

is either the smallest pair of words (u, v) such that $x\,u = y\,v$, if such a pair exists, or $\bot$ otherwise. Note that if $\mathsf{LD}(\mathsf{out}(\rho), \mathsf{out}(\rho')) \neq \bot$, then this is a pair where at least one of the two components is the empty word ε, and it is $(\varepsilon, \varepsilon)$ iff ρ and ρ' produce the same output. We also define the *lag* of (ρ, ρ') as the number

$$\Delta(\rho, \rho') \ =^{\mathrm{def}} \ \max_{\rho = \alpha\,\beta,\ \rho' = \alpha'\beta',\ |\alpha| = |\alpha'|} \ \big| \mathsf{LD}\big(\mathsf{out}(\alpha), \mathsf{out}(\alpha')\big) \big|$$

(by convention, we let $|\bot| = \infty > n$ for all $n \in \mathbb{N}$). Intuitively, $\Delta(\rho, \rho')$ describes how much the partial outputs produced by some prefixes of ρ and ρ' of the same length have diverged.

The proof of the decomposition theorem consists in arguing as follows.

The main combinatorial property. Given a k-valued one-way transducer $\mathcal{T}$, one constructs a multi-transducer $\mathcal{T}^{k+1}$, whose successful runs can be seen as $(k + 1)$-tuples of successful runs of $\mathcal{T}$ spelling out the same input and possibly different outputs. One proves the following combinatorial property (see [18]): *for every successful run $\bar\rho = (\rho_1, \dots, \rho_{k+1})$ of $\mathcal{T}^{k+1}$, there exist distinct components ρ_i, ρ_j of $\bar\rho$ that not only agree on the output (this is necessary since $\mathcal{T}$ is k-valued), but also have $\Delta(\rho_i, \rho_j) \leq n_0$, where n_0 is a large enough constant computed from $\mathcal{T}^{k+1}$.*

The idea underlying this result is that if two runs ρ, ρ' of $\mathcal{T}$ on the same input have produced at some point partial outputs that are sufficiently far from each other, i.e. $\Delta(\alpha, \alpha') > n_0$ for some prefixes α, α' of ρ, ρ', then this difference can be augmented so much that it cannot be overtaken by the suffixes of ρ, ρ' that come after α, α' — this gives two modified runs that produce different outputs. As $\mathcal{T}$ is k-valued, if one considers $k + 1$ pairs of runs, then at least two of them must produce the same output and have lag at most n_0.

The lag separation covering. Let Q be the state space of $\mathcal{T}$ and $<$ a lexicographical order (induced by a total order on transitions) between pairs of runs of $\mathcal{T}$ with the same input, i.e. $\rho < \rho'$ implies ρ, ρ' have the same input. One can construct a covering $\mathcal{U}_{n_0}$ of $\mathcal{T}$, whose states are the pairs (q, o), with $q \in Q$ and $o : Q \to 2^{\Gamma^* \times \Gamma^*}$, such that for all initial runs ρ of $\mathcal{U}_{n_0}$ ending in (q, o) and all states $r \in Q$,

$$\mathsf{out}(\rho) \ = \ \mathsf{out}(\rho|_1) \qquad \begin{array}{l} \text{(namely, the output produced by the run } \rho \text{ of } \mathcal{U}_{n_0} \\ \text{is the same as that of the underlying run } \rho|_1 \text{ of } \mathcal{T}) \end{array}$$

$$o(r) \ = \ \Big\{ \mathsf{LD}\big(\mathsf{out}(\rho|_1), \mathsf{out}(\rho')\big) \ : \ \begin{array}{l} \rho' \text{ initial run of } \mathcal{T} \text{ ending in } r \\ \text{such that } \rho' < \rho|_1 \text{ and } \Delta(\rho|_1, \rho) \leq n_0 \end{array} \Big\}$$

(note that the above pairs $\mathsf{LD}(\mathsf{out}(\rho|_1), \mathsf{out}(\rho'))$ have size at most n_0, so the transducer $\mathcal{U}_{n_0}$ is finite; for a proof of how these pairs can be maintained in the transitions of $\mathcal{U}_{n_0}$ see [15]).

From $\mathcal{U}_{n_0}$, one can then construct a transducer $\mathcal{V}_{n_0}$ whose successful runs with input w and output x simulate precisely the *least* runs among the successful ones of $\mathcal{T}$ with input w and output x. Formally, $\mathcal{V}_{n_0}$ is obtained from $\mathcal{U}_{n_0}$ by restricting the set of final states to those pairs (q, o) such that q is final in $\mathcal{T}$ and $(\varepsilon, \varepsilon) \notin o(r)$ for all $r \in Q$. Indeed, we have:

1. If ρ is a successful run of $\mathcal{V}_{n_0}$, then $\rho|_1$ is a successful run of $\mathcal{T}$ and for all other successful runs ρ' of $\mathcal{T}$ with the same input and the same output, either $\rho|_1 < \rho'$ or $\Delta(\rho|_1, \rho) > n_0$.
2. For all pairs (w, x) in the transduction defined by $\mathcal{T}$, if ρ' is the *least* successful run of $\mathcal{T}$ with input w and output x, then there is a successful run ρ of $\mathcal{V}_{n_0}$ such that $\rho' = \rho|_1$.

This shows that $\mathcal{V}_{n_0}$ is equivalent to $\mathcal{T}$.

In addition, the transducer $\mathcal{V}_{n_0}$ is *input-k-ambiguous*. This relies on the choice of n_0 and can be shown by way of contradiction as follows. Suppose that $\mathcal{V}_{n_0}$ admits $k + 1$ distinct successful runs $\rho_1, \dots, \rho_{k+1}$ on the same input w. By the combinatorial property stated

above, there exist two indices $i \neq j$ such that $\rho_i|_1$ and $\rho_j|_1$ have the same input, the same output, and satisfy $\Delta(\rho_i|_1, \rho_j|_1) \leq n_0$. However, this would contradict property 1. above.

The multi-skimming covering. The last part of the proof is fully generic and consists in decomposing a k-ambiguous automaton $\mathcal{A}$ (e.g. the input automaton of $\mathcal{V}_{n_0}$) into a finite union of unambiguous automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$ (the construction can be easily lifted to the transducer $\mathcal{V}_{n_0}$ by copying the outputs generated by the transitions).

Let $\mathcal{A}$ be a k-ambiguous automaton with state space Q. One first defines a covering $\mathcal{B}$ of $\mathcal{A}$ whose states are of the form (q, o), with $q \in Q$ and $o : Q \to \{0, \ldots, k-1\}$, such that for all initial runs of $\mathcal{B}$ ending in (q, o) and all states r,

$$o(r) = \left| \{ \rho' : \rho' \text{ initial run of } \mathcal{A} \text{ ending in } r \text{ such that } \rho' < \rho|_1 \} \right|$$

(see again [15] for a construction that maintains the function o along the transitions of $\mathcal{B}$).

Finally, one constructs some automata $\mathcal{B}_0, \ldots, \mathcal{B}_{k-1}$ that differ from $\mathcal{B}$ only in the sets of final states. Formally, for each $i = 0, \ldots, k-1$, one declares a state (q, o) final in $\mathcal{B}_i$ iff $q \in F$ and $\sum_{r \in F} o(r) = i$, where F is the set of final states of $\mathcal{A}$. Note that an initial run ρ of the automaton $\mathcal{B}_i$ is successful iff $\rho|_1$ is successful for $\mathcal{A}$ and there are exactly i successful runs of $\mathcal{A}$ that precede $\rho|_1$ in the lexicographic ordering. In particular, the automata $\mathcal{B}_0, \ldots, \mathcal{B}_{k-1}$ are all unambiguous and their union is equivalent to $\mathcal{A}$.

5 The proof of the conjecture for 1-register SSTs

In this section we prove a decomposition theorem for finite-valued SSTs with a single register:

▶ **Theorem 5.** *k-valued 1-register SSTs are effectively equivalent to finite unions of functional 1-register SSTs.*

From the sketch of the proof of the decomposition theorem for one-way transducers we see that there are two important results that need to be generalized in order to lift the theorem to SSTs. The first of them is the combinatorial property related to the lag $\Delta(\rho, \rho')$. A noticeable difference is that, in the presence of an SST $\mathcal{T}$ with 1 register, partial outputs are constructed by adding symbols both to the right and to the left of the register of $\mathcal{T}$. This means that, to correctly measure the distance between the register contents x, x' of runs, we need to take into account the possible contexts that can be added to both endpoints of x and x'. We thus modify the definition of *lead or delay* of a pair (x, x'):

$$\mathsf{LD}(x, x') =^{\text{def}} \{ (u, v, u', v') : u\,x\,v = u'\,x'\,v', (u = \varepsilon) \vee (u' = \varepsilon), (v = \varepsilon) \vee (v' = \varepsilon) \}.$$

The size of a tuple $\lambda = (u, v, u', v') \in \mathsf{LD}(x, x')$ is $|\lambda| = |u| + |u'| + |v| + |v'|$.

Let $\mathsf{reg}(\rho)$ denote the register content at the end of an initial run ρ of $\mathcal{T}$. Note that this is precisely the output produced by ρ if ρ is successful. Intuitively, a tuple $\lambda \in \mathsf{LD}(\mathsf{reg}(\rho), \mathsf{reg}(\rho'))$ describes a possible alignment between the register contents at the end of ρ and ρ'. Such a tuple has one of the following forms (the annotations under the tuples describe succinctly the type of alignment between $\mathsf{reg}(\rho)$ and $\mathsf{reg}(\rho')$):

Accordingly, we define the *lag* of a pair of initial runs (ρ, ρ') as the number

$$\Delta(\rho, \rho') =^{\mathrm{def}} \max_{\rho = \alpha\,\beta,\, \rho' = \alpha'\beta',\, |\alpha| = |\alpha'|} \min_{\lambda \in \mathsf{LD}(\mathsf{reg}(\alpha), \mathsf{reg}(\alpha'))} |\lambda|$$

(as usual, we assume that $\min \varnothing = \infty > n$ for all $n \in \mathbb{N}$).

The combinatorial property. Along the same line of the proof in Section 4, given a k-valued SST $\mathcal{T}$ with 1 register, we construct the SST $\mathcal{T}^{k+1}$ with $k+1$ registers that simulates $k+1$ copies of $\mathcal{T}$ running in parallel on the same input. Given a run $\bar{\rho} = (\rho_1, \ldots, \rho_{k+1})$ of $\mathcal{T}^{k+1}$, we denote by $\bar{\rho}|_i$ the i-th run ρ_i of $\bar{\rho}$. To correctly work with $\mathcal{T}^{k+1}$, we need to assume that $\mathcal{T}$ has been normalized in such a way that the register content can only grow. Formally, we say that $\mathcal{T} = (\Sigma, \Gamma, Q, \{x\}, x, U, I, E, F)$ is *normalized* if for all register updates $f \in U$, $f(x)$ contains exactly one occurrence of x. It is easy to transform any 1-register SST into an equivalent normalized SST.

▸ **Lemma 6.** *One can normalize any 1-register SST.*

We now turn towards the main combinatorial property:

▸ **Proposition 7.** *Given a normalized k-valued SST $\mathcal{T}$, one can compute $n_0 \in \mathbb{N}$ such that, for all successful runs $\bar{\rho}$ of $\mathcal{T}^{k+1}$, $\mathsf{reg}(\rho|_i) = \mathsf{reg}(\rho|_j)$ and $\Delta(\rho|_i, \rho|_j) \le n_0$ for some $i \ne j$.*

Recall that the technique from [18] for proving the combinatorial result for one-way transducers consists in augmenting the differences between pairs of runs of $\bar{\rho}$, until the runs are shown to produce pairwise distinct outputs. In the case of SSTs, it becomes difficult to apply such a technique. Instead, we rely on a powerful result of Kortelainen [13] which has been later improved and simplified by Saarela (Theorem 9 below) and that is related to solutions of word equations with iterated factors. Word equations will indeed be obtained by comparing the outputs produced by pumped versions of successful runs of $\mathcal{T}$. Formally, we call *loop* of a run ρ any factor of ρ that starts and ends in the same state. A loop of ρ is *maximal* if it is not strictly contained in any other loop of ρ. Given a set L of pairwise non-overlapping loops of ρ, we denote by $\mathsf{pump}_L^h(\rho)$ the run obtained from ρ by repeating h times the loops in the set L (if $h = 0$ this amounts at removing the loops). The following lemma is simple but crucial for our result (notice that it does not hold in general for SSTs with more than one register).

▸ **Lemma 8.** *Given an initial run ρ of a normalized 1-register SST and a set L of pairwise non-overlapping loops of ρ, one can factorize $\mathsf{reg}(\rho)$ as $u_0 v_1 u_1 \ldots u_{2m-1} v_{2m} u_{2m}$ in such a way that, for all $h \in \mathbb{N}$, $\mathsf{reg}\big(\mathsf{pump}_L^h(\rho)\big) = u_0 (v_1)^h u_1 \ldots u_{2m-1} (v_{2m})^h u_{2m}$.*

▸ **Theorem 9** (Theorem 4.3 in [14]). *The set of solutions of an equation of the form*

$$u_0 (v_1)^h u_1 \ldots u_{m-1} (v_m)^h u_m = u'_0 (v'_1)^h u'_1 \ldots u_{m-1} (v'_m)^h u'_m$$

where h is the unknown, is either $\mathbb{N}$ or a finite subset of $\mathbb{N}$.

Since the loops of any run $\bar{\rho}$ of $\mathcal{T}^{k+1}$ can be equally seen as loops of the runs $\rho|_1, \ldots, \rho|_{k+1}$ of $\mathcal{T}$, we can write $\mathsf{pump}_L^h(\rho|_i)$ whenever L is a set of pairwise non-overlapping loops of $\bar{\rho}$. The following property is an immediate consequence of Lemma 8 and Theorem 9:

▸ **Corollary 10.** *For every successful run $\bar{\rho}$ of $\mathcal{T}^{k+1}$, every set L of pairwise non-overlapping loops of $\bar{\rho}$, and every pair of indices $i, j \in \{1, \ldots, k+1\}$,*

$$\mathsf{reg}(\rho|_i) \ne \mathsf{reg}(\rho|_j) \quad implies \quad \exists h_0 \in \mathbb{N} \; \forall h \ge h_0 \quad \mathsf{reg}\big(\mathsf{pump}_L^h(\rho|_i)\big) \ne \mathsf{reg}\big(\mathsf{pump}_L^h(\rho|_j)\big).$$

We are now ready to prove Proposition 7 by way of contradiction, that is, we fix a large enough n_0 (see Lemma 11 below for the precise value of n_0) and we assume that there is a successful run $\bar{\rho}$ of $\mathcal{T}^{k+1}$ such that, for all i, j, $\mathsf{reg}(\rho|_i) = \mathsf{reg}(\rho|_j)$ implies $\Delta(\rho|_i, \rho|_j) > n_0$, and we show that $\mathcal{T}$ outputs at least $k + 1$ different words on the same input w, a contradiction with the k-valuedness of $\mathcal{T}$. We will do so by exploiting an induction on the number of pairs of runs $\rho|_i, \rho|_j$ with the same output. For the sake of brevity, we denote by $c_\mathcal{T}$ the *capacity* of $\mathcal{T}$, that is, the maximum number of symbols that are added to the content of the register during a single transition of $\mathcal{T}$. We further define $E(\bar{\rho}) = \big\{(i,j) \in \{1, \ldots, k+1\} : \mathsf{reg}(\rho|_i) = \mathsf{reg}(\rho|_j)\big\}$. We prove the following invariant, which leads towards a contradiction of k-valuedness of $\mathcal{T}$:

▸ **Lemma 11.** *If there is a successful run $\bar{\rho}$ of $\mathcal{T}^{k+1}$ with $E(\bar{\rho}) \neq \varnothing$ and*

$$\forall i, j \qquad (i,j) \in E(\bar{\rho}) \qquad implies \qquad \Delta(\rho|_i, \rho|_j) > c_\mathcal{T} \cdot |Q|^{k+1} \qquad\qquad (\star)$$

then there is another such run $\bar{\rho}'$ of $\mathcal{T}^{k+1}$ satisfying $E(\bar{\rho}') \subsetneq E(\bar{\rho})$ and $(\star)$.

Proof. For every pair $(i,j) \in E(\bar{\rho})$, we can split the run $\bar{\rho}$ at the first point where the lag exceeds the value $c_\mathcal{T} \cdot |Q|^{k+1}$, that is, we can factorize $\bar{\rho}$ as $\bar{\alpha}_{i,j} \, \bar{\alpha}'_{i,j}$ in such a way that $\bar{\alpha}_{i,j}$ is the shortest prefix satisfying $\Delta(\bar{\alpha}_{i,j}|_i, \bar{\alpha}_{i,j}|_j) > c_\mathcal{T} \cdot |Q|^{k+1}$. Based on this, we choose a pair $(i,j) \in E(\bar{\rho})$ that induces a factorization of $\bar{\rho}$ with the longest prefix $\bar{\alpha}_{i,j}$. Clearly, for all other pairs $(i',j') \in E(\bar{\rho})$, $\bar{\alpha}_{i',j'}$ is a prefix of $\bar{\alpha}_{i,j}$, and hence

$$\Delta\big(\bar{\alpha}_{i,j}|_{i'}, \bar{\alpha}_{i,j}|_{j'}\big) \;\geq\; \Delta\big(\bar{\alpha}_{i',j'}|_{i'}, \bar{\alpha}_{i',j'}|_{j'}\big) \;=\; \Delta\big(\bar{\rho}|_{i'}, \bar{\rho}|_{j'}\big) \;>\; c_\mathcal{T} \cdot |Q|^{k+1}$$

Now that the pair (i,j) is fixed, we further factorize the suffix $\bar{\alpha}'_{i,j}$ of $\bar{\rho}$ as follows:

$$\bar{\alpha}'_{i,j} \;=\; \bar{\beta}_0 \, \bar{\gamma}_1 \, \bar{\beta}_1 \, \ldots \, \bar{\beta}_{m-1} \, \bar{\gamma}_m \, \bar{\beta}_m$$

where $L = \{\bar{\gamma}_1, \ldots, \bar{\gamma}_m\}$ is a maximal set of pairwise non-overlapping maximal loops of $\bar{\alpha}'_{i,j}$. Further let $\bar{\rho}^{(h)} = \mathsf{pump}_L^h(\bar{\rho})$, for all $h \in \mathbb{N}$, and observe that $\bar{\rho}^{(1)} = \bar{\rho}$.

We claim that there exists $h_{i,j} \in \mathbb{N}$ such that for all $h \geq h_{i,j}$, $\mathsf{reg}(\bar{\rho}^{(h)}|_i) \neq \mathsf{reg}(\bar{\rho}^{(h)}|_j)$. Indeed, if this were not the case, then there would exist infinitely many $h \in \mathbb{N}$ such that $\mathsf{reg}(\bar{\rho}^{(h)}|_i) = \mathsf{reg}(\bar{\rho}^{(h)}|_j)$. In particular, by Theorem 9, we would have $\mathsf{reg}(\bar{\rho}^{(h)}|_i) = \mathsf{reg}(\bar{\rho}^{(h)}|_j)$ for $h = 0$. Note that $\bar{\alpha}_{i,j}$ is also prefix of $\bar{\rho}^{(0)}$ and the remaining part of $\bar{\rho}^{(0)}$ has length at most $|Q|^{k+1}$, since otherwise it would contain a loop which is either contained in one of the $\bar{\beta}_i$'s, in which case we would get a contradiction with the maximality of L, or is spread between several $\bar{\beta}_i$'s, which would then be in contradiction with the maximality of the loops γ_i. Now, consider the register contents at the end of the i-th and j-th runs of $\bar{\alpha}_{i,j}$ and $\bar{\rho}^{(0)}$; that is, let $x = \mathsf{reg}(\bar{\alpha}_{i,j}|_i)$, $x' = \mathsf{reg}(\bar{\alpha}_{i,j}|_j)$, $x_0 = \mathsf{reg}(\bar{\rho}^{(0)}|_i)$, $x'_0 = \mathsf{reg}(\bar{\rho}^{(0)}|_j)$. Since $\mathcal{T}$ is normalized, there exist some words u, v, u', v' such that $x_0 = u\,x\,v$ and $x'_0 = u'x'v'$. Moreover, since these words represent the content that is added to the registers by the updates performed along a suffix of $\bar{\rho}^{(0)}$ of length at most $|Q|^{k+1}$, we know that uv and $u'v'$ have length at most $c_\mathcal{T} \cdot |Q|^{k+1}$. Since $x_0 = \mathsf{reg}(\bar{\rho}^{(0)}|_i) = \mathsf{reg}(\bar{\rho}^{(h)}|_j) = x'_0$, we know that $(u, v, u', v') \in \mathsf{LD}(x, x')$. But, by definition, $\Delta\big(\bar{\alpha}_{i,j}|_i, \bar{\alpha}_{i,j}|_j\big) \;=\; \min\big\{|\lambda| : \lambda \in \mathsf{LD}(x,x')\big\} \;\leq\; c_\mathcal{T} \cdot |Q|^{k+1}$ and therefore, $\Delta\big(\bar{\alpha}_{i,j}|_i, \bar{\alpha}_{i,j}|_j\big) \leq |u| + |v| + |u'| + |v'| \leq c_\mathcal{T} |Q|^{k+1}$ which contradicts the definition of $\bar{\alpha}_{i,j}$.

We have just proved that there is $h_{i,j} \in \mathbb{N}$ such that, for all $h \geq h_{i,j}$, $(i,j) \notin E(\bar{\rho}^{(h)})$. Consider now the other pairs $(i',j') \in \{1, \ldots, k+1\}^2 \smallsetminus E(\bar{\rho})$, namely, such that $\mathsf{reg}(\bar{\rho}|_{i'}) \neq \mathsf{reg}(\bar{\rho}|_{j'})$. By Corollary 10, for every such pair (i',j') there is $h_{i',j'} \in \mathbb{N}$ such that, for all $h \geq h_{i',j'}$, $\mathsf{reg}\big(\bar{\rho}^{(h)}|_{i'}\big) \neq \mathsf{reg}\big(\bar{\rho}^{(h)}|_{j'}\big)$. We can thus define $\bar{\rho}' = \bar{\rho}^{(h')}$, where h' is the maximum of the $h_{i',j'}$'s for all $(i',j') \notin E(\bar{\rho})$. In this way we get $(i,j) \notin E(\bar{\rho}')$ and $E(\bar{\rho}') \subsetneq E(\bar{\rho})$. Finally, it remains to verify that $\bar{\rho}'$ satisfies the condition $(\star)$. Consider a pair $(i',j') \in E(\bar{\rho}')$. Since $\bar{\alpha}_{i',j'}$ is a prefix of $\bar{\rho}'$ and $\Delta\big(\bar{\alpha}_{i,j}|_i, \bar{\alpha}_{i,j}|_j\big) > c_\mathcal{T} \cdot |Q|^{k+1}$, we conclude that $\Delta\big(\bar{\rho}'|_i, \bar{\rho}'|_j\big) > c_\mathcal{T} \cdot |Q|^{k+1}$ as well. ◂

The lag separation covering.   The second and last point that we need to explain is the construction of a lag separation covering of $\mathcal{T}$, that is, a 1-register SST $\mathcal{U}_{n_0}$ with the following properties. The states of $\mathcal{U}_{n_0}$ are the pairs (q, o), with $q \in Q$, Q state space of $\mathcal{T}$, and o some stored information determining a certain function $\hat{o} : Q \to 2^{\Gamma^* \times \Gamma^* \times \Gamma^* \times \Gamma^*}$ (we will see soon why o cannot be directly defined as the function $\hat{o}$ itself). For all initial runs ρ of $\mathcal{U}_{n_0}$ ending in (q, o) and all states $r \in Q$, we must have

$$\mathsf{reg}(\rho) \;=\; \mathsf{reg}(\rho|_1) \qquad \begin{array}{l}\text{(namely, the register content at the end of the run } \rho \text{ of } \mathcal{U}_{n_0}\\ \text{is the same as that of the underlying run } \rho|_1 \text{ of } \mathcal{T})\end{array}$$

$$\hat{o}(r) \;=\; \bigcup \Big\{ \mathsf{LD}\big(\mathsf{reg}(\rho|_1), \mathsf{reg}(\rho')\big) \;:\; \begin{array}{l}\rho' \text{ initial run of } \mathcal{T} \text{ ending in } r \\ \text{such that } \rho' < \rho|_1 \text{ and } \Delta(\rho|_1, \rho) \le n_0\end{array} \Big\}.$$

One can then transform $\mathcal{U}_{n_0}$ into an input-k-ambiguous SST $\mathcal{V}_{n_0}$ equivalent to $\mathcal{T}$: this is done as explained in Section 4 by restricting the set of final states to those pairs (q, o) such that q is final in $\mathcal{T}$ and $(\varepsilon, \varepsilon, \varepsilon, \varepsilon) \notin \hat{o}(r)$ for all $r \in Q$. The proof that $\mathcal{V}_{n_0}$ is input-k-ambiguous relies on the combinatorial property for normalized 1-register SSTs that we gave before. Finally, one applies the multi-skimming covering construction to $\mathcal{V}_{n_0}$ in order to obtain k input-unambiguous SSTs $\mathcal{T}_0, \dots, \mathcal{T}_{k-1}$ whose union is equivalent to $\mathcal{T}$.

For the sake of brevity, let $\Lambda^{\le n_0}$ be the set of tuples $\lambda = (u, v, u', v')$ of size at most n_0, and let $\mathsf{LD}^{\le n_0}(x, x') = \mathsf{LD}(x, x') \cap \Lambda^{\le n_0}$. The construction of $\mathcal{U}_{n_0}$ boils down to showing that one can maintain the information about the set $\mathsf{LD}^{\le n_0}(\mathsf{reg}(\rho), \mathsf{reg}(\rho'))$ while processing any two runs ρ, ρ' of $\mathcal{T}$, provided that $\Delta(\rho, \rho') \le n_0$. Indeed, suppose that, by some means, we are able to construct a deterministic automaton $\mathcal{A}$, with state space S, together with a function $\hat{} : S \to 2^{\Lambda^{\le n_0}}$ such that, whenever $\mathcal{A}$ reads an initial run (ρ, ρ') of $\mathcal{T}^2$ (seen as a sequence of pairs of transition rules of $\mathcal{T}$), then $\mathcal{A}$ reaches a state $s_{\rho, \rho'}$ such that $\hat{s}_{\rho, \rho'}$ is either the set $\mathsf{LD}^{\le n_0}(\mathsf{reg}(\rho), \mathsf{reg}(\rho'))$ or the empty set, depending on whether $\Delta(\rho, \rho') \le n_0$ or not. Using the automaton $\mathcal{A}$ one can construct the covering $\mathcal{U}_{n_0}$ as follows:

- The states of $\mathcal{U}_{n_0}$ are the pairs (q, o), with $q \in Q$ and $o \in S \times (2^S)^Q$. The component q is for simulating an initial run ρ of $\mathcal{T}$ on the given input. The stored information o consists of two parts: a state $s \in S$ for simulating the run of $\mathcal{A}$ on (ρ, ρ) and a function $t : Q \to 2^S$ for simulating the possible runs of $\mathcal{A}$ on inputs of the form (ρ, ρ'), for all initial runs ρ' of $\mathcal{T}$ such that $\rho' < \rho$, with a partitioning based on the last state of ρ'. The latter part t also determines the function $\hat{o} : Q \to 2^{\Lambda^{n_0}}$ associated with the stored information, that is, $\hat{o}(r) = \bigcup_{s \in t(r)} \hat{s}$ for all $r \in Q$.
- The possible transitions of $\mathcal{U}_{n_0}$ are of the form $(q_1, o_1) \xrightarrow{a/f}_{\mathcal{U}_{n_0}} (q_2, o_2)$, with $o_1 = (s_1, t_1)$ and $o_2 = (s_2, t_2)$, whenever $\tau = (q_1, a, f, q_2)$ is a transition rule of $\mathcal{T}$, $(s_1, (\tau, \tau), s_2)$ is a transition rule of $\mathcal{A}$, and, for all $r_2 \in Q$,

$$t_2(r_2) \;=^{\mathrm{def}}\; \Big\{ s' \;:\; \begin{array}{l}(s_1, (\tau, \tau'), s') \text{ transition rule of } \mathcal{A}, \\ \tau' = (q_1, a, f', r_2) \text{ transistion rule of } \mathcal{T}, \quad \tau' < \tau\end{array} \Big\}$$

$$\cup \Big\{ s' \;:\; \begin{array}{ll}s \in t_1(r_1), & (s, (\tau, \tau'), s') \text{ transition rule of } \mathcal{A}, \\ r_1 \in Q, & \tau' = (r_1, a, f', r_2) \text{ transition rule of } \mathcal{T}\end{array} \Big\}.$$

- The initial states of $\mathcal{U}_{n_0}$ are the pairs (q, o_0), with $o_0 = (s_0, t_0)$, q initial state of $\mathcal{T}$, s_0 initial state of $\mathcal{A}$, and $t_0(r) = \varnothing$ for all $r \in Q$. Similarly, the final states of $\mathcal{U}_{n_0}$ are the pairs (q, o), with q final state of $\mathcal{T}$.

Maintaining alignments under updates.   After the previous argument, we are left to outline the construction of an automaton $\mathcal{A}$ that recognizes pairs of runs (ρ, ρ') and from the states of which we can extract $\mathsf{LD}^{\le n}(\mathsf{reg}(\rho), \mathsf{reg}(\rho'))$ when $\Delta(\rho, \rho') \le n$,for some fixed
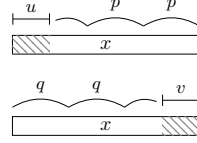
number n. More specifically, we can focus on the following sub-goal: for a fixed $n \in \mathbb{N}$, maintain a suitable data structure in bounded memory (the set of these structures form the state space of $\mathcal{A}$) that determines $\mathsf{LD}^{\leq n}(x, x')$ while the registers x, x' are being updated, as usual under the proviso that this set remains non-empty all along the computation. A possible way to attain this sub-goal is to try to compute the set $\mathsf{LD}^{\leq n}(s\,x\,t, s'x't')$ directly from a given set $\mathsf{LD}^{\leq n}(x, x')$ and some given words s, t, s', t'. Note that the actual content of the registers x, x' is not part of the input of this computational problem. Along this idea, we disclose the dependencies of the sets $\mathsf{LD}^{\leq n}(xa, x')$ and $\mathsf{LD}^{\leq n}(ax, x')$ from $\mathsf{LD}^{\leq n+1}(x, x')$:

$$
\begin{aligned}
\mathsf{LD}^{\leq n}(xa, x') \ = \ & \big\{ \, (u, \varepsilon, u', \boldsymbol{v'a}) \ : \ (u, \varepsilon, u', \boldsymbol{v'}) \in \mathsf{LD}^{\leq n-1}(x, x') \, \big\} \\
& \cup \big\{ \, (u, \boldsymbol{v}, u', \varepsilon) \ : \ (u, \boldsymbol{av}, u', \varepsilon) \in \mathsf{LD}^{\leq n+1}(x, x') \, \big\}
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
\mathsf{LD}^{\leq n}(ax, x') \ = \ & \big\{ \, (\varepsilon, v, \boldsymbol{au'}, v') \ : \ (\varepsilon, v, \boldsymbol{u'}, v') \in \mathsf{LD}^{\leq n-1}(x, x') \, \big\} \\
& \cup \big\{ \, (\boldsymbol{u}, v, \varepsilon, v') \ : \ (\boldsymbol{ua}, v, \varepsilon, v') \in \mathsf{LD}^{\leq n+1}(x, x') \, \big\}.
\end{aligned}
\tag{2}
$$

Using these dependencies, one can describe any set of the form $\mathsf{LD}^{\leq n}(\tilde{x}, \tilde{x}')$, where $\tilde{x} = s\,x\,t$ and $\tilde{x}' = s'x't'$, on the basis of the set $\mathsf{LD}^{\leq \tilde{n}}(x, x')$ and the words s, t, s', t', where $\tilde{n} = n + |s| + |t| + |s'| + |t'|$. However, the fact that $\tilde{n}$ may be larger than n seems to require a knowledge of alignments of arbitrarily large size, which we cannot assume. Luckily we can overcome this problem by exploiting the condition that $\mathsf{LD}^{\leq n}(x, x') \neq \varnothing$ and by using an auxiliary data structure for maintaining some knowledge about the periodicities in x and x'.

We say that a word x has period $k \in \mathbb{N}$ if $x(i) = x(i+k)$ for all $1 \leq i \leq |x| - k$. For example, *abcab* has period 3, but also period $5, 6, 7$, etc. We define the two sets

$$
R(x) \ \stackrel{\text{def}}{=} \ \big\{ (u, p) \ : \ x = u\,y \text{ and } y\,p \text{ is periodic with period } |p| \big\}
$$

$$
L(x) \ \stackrel{\text{def}}{=} \ \big\{ (v, q) \ : \ x = y\,v \text{ and } q\,y \text{ is periodic with period } |q| \big\}
$$

Intuitively, $(u, p) \in R(x)$ iff x begins with u and continues with a word y whose period $|p|$ is preserved under extensions with p to the right, and similarly for $(v, q) \in L(x)$ (see the figure). As usual, we let $R^{\leq m}(x)$ (resp. $L^{\leq m}(x)$) be the restriction of $R(x)$ (resp. $L(x)$) to the pairs of size at most m (the size being the sum of the lengths of the words in the pair).

For any fixed $m \in \mathbb{N}$, it is possible to devise a bounded-memory data structure that can be maintained while the register x is being updated and that, at every moment, determines, the sets $R^{\leq m}(x)$ and $L^{\leq m}(x)$. The data structure basically recalls the following information:

- the exact content of x when its length is smaller than m,
- the unique pair of words $(u, p) \in R^{\leq m}(x)$ (if it exists) that has the smallest length $|u| + |p|$,
- the unique pair of words $(v, q) \in L^{\leq m}(x)$ (if it exists) that has the smallest length $|v| + |q|$.

From now on we assume that the sets $R^{\leq m}(x), L^{\leq m}(x), R^{\leq m}(x'), L^{\leq m}(x')$ are known during any sequence of updates of x and x'.

Below we show that any non-empty set $\mathsf{LD}^{\leq n}(x, x')$, paired with the information given by the sets $R^{\leq \tilde{n}}(x), L^{\leq \tilde{n}}(x), R^{\leq \tilde{n}}(x'), L^{\leq \tilde{n}}(x')$, fully determines the content of $\mathsf{LD}^{\leq \tilde{n}}(x, x')$. Thanks to the dependencies given in Equations (1) and (2), this is sufficient for computing also the set $\mathsf{LD}^{\leq n}(\tilde{x}, \tilde{x}')$, where $\tilde{x}$ (resp. $\tilde{x}'$) is obtained from x (resp. x') by prepending/appending at most $\tilde{n} - n$ symbols.

We define two sets, $G_{n,m}(x, x')$ and $H_{n,m}(x, x')$, on the basis of the sets $\mathsf{LD}^{\leq n}(x, x')$, $L^{\leq m}(x), R^{\leq m}(x), L^{\leq m}(x'), R^{\leq m}(x')$. The elements of $G_{n,m}(x, x')$ represent possible tuples of alignment type $\overline{}$:

$$G_{n,m}(x,x') = \left\{ (\varepsilon, \boldsymbol{p}\boldsymbol{v}, \boldsymbol{u'}\boldsymbol{q}, \varepsilon) \; : \; (\varepsilon, \boldsymbol{v}, \boldsymbol{u'}, \varepsilon) \in \mathsf{LD}^{\leq n}(x,x'), \; \begin{array}{l} (u', \boldsymbol{p}) \in P^{\leq m}(x), \\ (v, \boldsymbol{q}) \in Q^{\leq m}(x'), \end{array} \; |p| = |q| \right\}$$

$$\cup \left\{ (\varepsilon, \boldsymbol{v}, \boldsymbol{u'}, \varepsilon) \; : \; (u, \varepsilon, \varepsilon, v') \in \mathsf{LD}^{\leq n}(x,x'), \; \begin{array}{ll} (\varepsilon, p) \in P^{\leq m}(x), & p = \boldsymbol{v}\,v', \\ (\varepsilon, q) \in Q^{\leq m}(x'), & q = u\,\boldsymbol{u'}, \end{array} \; |p| = |q| \right\}$$

$$\cup \left\{ (\varepsilon, \boldsymbol{v}, \boldsymbol{u'}\boldsymbol{q}, \varepsilon) \; : \; (\varepsilon, \varepsilon, \boldsymbol{u'}, v') \in \mathsf{LD}^{\leq n}(x,x'), \; \begin{array}{l} (u', p) \in P^{\leq m}(x), \\ (\varepsilon, \boldsymbol{q}) \in Q^{\leq m}(x'), \end{array} \; p = \boldsymbol{v}\,v', \; |p| = |q| \right\}$$

$$\cup \left\{ (\varepsilon, \boldsymbol{p}\boldsymbol{v}, \boldsymbol{u'}, \varepsilon) \; : \; (u, \boldsymbol{v}, \varepsilon, \varepsilon) \in \mathsf{LD}^{\leq n}(x,x'), \; \begin{array}{l} (\varepsilon, \boldsymbol{p}) \in P^{\leq m}(x), \\ (v, q) \in Q^{\leq m}(x'), \end{array} \; q = u\,\boldsymbol{u'}, \; |p| = |q| \right\}.$$

The set $H_{n,m}(x,x')$ is defined in a symmetric way, as if the underlying order of the words were reversed. Accordingly, its elements represent possible tuples of alignment type $\rightleftharpoons$. For the sake of brevity, we omit the tedious definition of $H_{n,m}(x,x')$.

The following lemma gives a way to compute the set $\mathsf{LD}^{\leq \tilde{n}}(x,x')$ (and hence $\mathsf{LD}^{\leq n}(\tilde{x},\tilde{x}')$ for any $\tilde{x} = s\,x\,t$ and $\tilde{x}' = s'x't'$, with $|s| + |t| + |s'| + |t'| \leq \tilde{n} - n$): for this it suffices to construct $\mathsf{LD}^{\leq n}(x,x') \cup G_{n,n+\tilde{n}}(x,x') \cup H_{n,n+\tilde{n}}(x,x')$ and then remove from it the tuples of size larger than $\tilde{n}$. Thanks to the lemma, this results in the set $\mathsf{LD}^{\leq \tilde{n}}(x,x')$.

▸ **Lemma 12.** *For all $n \leq \tilde{n}$, if $\mathsf{LD}^{\leq n}(x,x')$ is non-empty, then*

$$\mathsf{LD}^{\leq \tilde{n}}(x,x') \; \subseteq \; \mathsf{LD}^{\leq n}(x,x') \cup G_{n,n+\tilde{n}}(x,x') \cup H_{n,n+\tilde{n}}(x,x') \; \subseteq \; \mathsf{LD}(x,x').$$

Tuples $(\mathsf{LD}^{\leq n}(x,x'), L^{\leq 2n+2c\tau}(x), R^{\leq 2n+2c\tau}(x), L^{\leq 2n+2c\tau}(x'), R^{\leq 2n+2c\tau}(x'))$ thus form an adequate state space for $\mathcal{A}$ when x and x' are the contents of the registers of the runs $\mathcal{A}$ is recognizing. This completes the proof of Theorem 5. And, as for Corollary 2, we obtain:

▸ **Corollary 13.** *One can decide if two k-valued 1-register SSTs $\mathcal{T}, \mathcal{T}'$ are equivalent.*

6   Conclusions

We have proven a decomposition theorem for finite-valued 1-register SSTs, generalizing a similar result of Weber for one-way transducers. This result constitutes a milestone towards a decomposition theorem for finite-valued SSTs with multiple registers paving the way to the decidability of the equivalence problem, and to a correspondence with finite-valued two-way transducers. This way meets however two difficulties.

The first difficulty is related to establishing the analogous of Proposition 7. Indeed, the latter proposition relies on the exact correspondence between the number of iterations of loops in a successful run of an SST and the number of iterations of certain factors in the produced output (cf. Lemma 8). This correspondence is broken in the presence of updates that concatenate different registers, namely, functions f such that, for some register x, $f(x)$ contains at least two occurrences of distinct registers. In particular, iterating a loop may not result in iterating factors in the output and, even worse, it can be the case that iterating loops make two different runs become equal for all iteration except a finite number of them. This is a situation which does not show up when we only deal with one register. In view of this, one is either likely to need a more complicated induction in order to prove the analogous of Proposition 7 for SSTs with multiple registers, or has to pertain to an argument about string combinatorics that is more general than that of Kortelainen and Sarelaa.

Another difficulty, which is perhaps even more challenging, lies in the construction of the lag separation covering of $\mathcal{T}$, and more precisely, in the problem of defining an appropriate notion of *lead or delay* between tuples of registers, and in maintaining this object in bounded memory while the registers are being updated. In particular, it seems that taking the obvious generalization of tuple measure we have taken for the 1-register case cannot be maintained using a finite memory.

References

1 R. Alur. Streaming string transducers. In *Proceedings of the 18th International Workshop on Logic, Language, Information and Computation (WoLLIC)*, volume 6642 of *LNCS*, page 1, 2011.

2 R. Alur and P. Cerny. Expressiveness of streaming string transducers. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 1–12, 2010.

3 R. Alur and P. Cerny. Streaming transducers for algorithmic verification of single-pass list processing programs. In *Proceedings of the 38th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2011.

4 R. Alur and J.V. Deshmukh. Nondeterministic streaming state transducers. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 6756 of *LNCS*, pages 1–20, 2011.

5 B. Courcelle. Monadic second-order graph transductions: a survey. *Theoretical Computer Science*, 126:53–75, 1994.

6 J. Engelfriet and H. Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, 2001.

7 J. Engelfriet and H.J. Hoogeboom. MSO-definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, 2001.

8 J. Engelfriet and S. Maneth. Macro Tree Transducers, Attribute Grammars, and MSO Definable Tree Translations. *Infomation and Computation*, 154(1):34–91, 1999.

9 E. Filiot and P.-A. Reynier. On streaming string transducers and HDT0L systems. *CoRR*, abs/1412.0537, 2014.

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

11 E.M. Gurari and O.H. Ibarra. A note on finite-valued and finitely ambiguous transducers. *Mathematical Systems Theory*, 16(1):61–66, 1983.

12 K. Culik II and J. Karhumäki. The equivalence of finite valued transducers (on hdt0l languages) is decidable. *Theoretical Computer Science*, 47:71–84, 1986.

13 J. Kortelainen. On the system of word equations $x_0 u_1^i x_1 u_2^i x_2 \dots u_m^i x_m = y_0 v_1^i y_1 v_2^i y_2 \dots v_m^i y_m$ $(i = 0, 1, 2, \dots)$ in a free monoid. *Journal of Automata, Languages and Combinatorics*, 3(1):43–57, 1998.

14 A. Saarela. Systems of word equations, polynomials and linear algebra: a new approach. *European Journal of Combinatorics*, 47(5):1–14, 2015.

15 J. Sakarovitch and R. De Souza. On the decomposition of k-valued rational relations. In *Proceedings of the 25th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2008.

16 J.C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.

17 A. Weber. On the valuedness of finite transducers. *Acta Informatica*, 27(8):749–780, 1990.

18 A. Weber. Decomposing a k-valued transducer into k unambiguous ones. *Informatique théorique et applications*, 30(5):379–413, 1996.