A Regular and Complete Notion of Delay for Streaming String Transducers

Emmanuel Filiot
Université libre de Bruxelles, Belgium

Ismaël Jecker
University of Warsaw, Poland

Christof Löding
RWTH Aachen University, Germany

Sarah Winter
Université libre de Bruxelles, Belgium

Abstract

The notion of delay between finite transducers is a core element of numerous fundamental results of transducer theory. The goal of this work is to provide a similar notion for more complex abstract machines: we introduce a new notion of delay tailored to measure the similarity between streaming string transducers (SST). We show that our notion is *regular*: we design a finite automaton that can check whether the delay between any two SSTs executions is smaller than some given bound. As a consequence, our notion enjoys good decidability properties: in particular, while equivalence between non-deterministic SSTs is undecidable, we show that equivalence *up to fixed delay* is decidable. Moreover, we show that our notion has good *completeness* properties: we prove that two SSTs are equivalent if and only if they are equivalent up to some (computable) bounded delay. Together with the regularity of our delay notion, it provides an alternative proof that SSTs equivalence is decidable. Finally, the definition of our delay notion is machine-independent, as it only depends on the *origin semantics* of SSTs. As a corollary, the completeness result also holds for equivalent machine models such as deterministic two-way transducers, or MSO transducers.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Streaming string transducers, Delay, Origin

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.32

Related Version *Full Version*: `https://arxiv.org/abs/2205.04287`

Funding This work was partially supported by the Fonds de la Recherche Scientifique – F.R.S.-FNRS under the MIS project F451019F.
Emmanuel Filiot: Emmanuel Filiot is a senior research associate at F.R.S-FNRS.
Sarah Winter: Sarah Winter is a postdoctoral researcher at F.R.S.-FNRS.

1 Introduction

Transducers are another fundamental extension of finite automata for computing *functions*, and more generally *relations*, between structures. In this paper, we consider string-to-string transducers, which operate on (input) strings and produce (output) strings. The most basic, *finite transducers*, are obtained by adding output words on the transitions of a finite automaton [32, 21]. At the heart of several important results in the theory of finite transducers is a notion, called *delay*, allowing to finely compare, in a regular way (with a finite automaton), transducer executions. The goal of this paper is to provide such a notion for a strictly more powerful class of transducers, streaming string transducers [1], which have

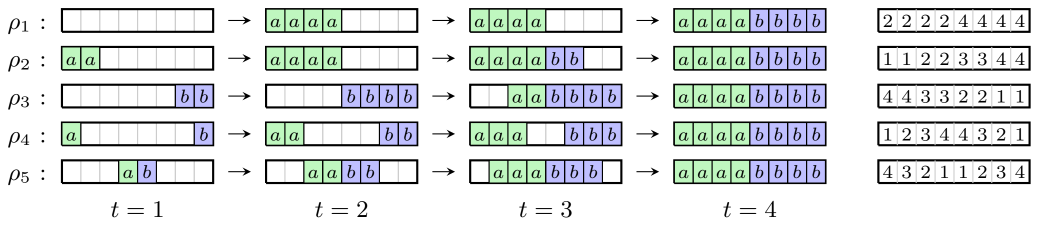


Figure 1 Five ways of producing the output sequence a^4b^4, and the corresponding origin functions.

received a lot of attention in the recent years, due to their robustness and equivalence with many other formalisms to define string-to-string functions. In particular, this paper answers positively the following high-level question:

Is it possible to measure in a regular manner how differently executions of equivalent streaming string transducers produce their outputs?

Finite transducers. We first explain how the above question is answered for finite transducers. Transitions of finite transducers over some alphabet Σ are tuples (p, σ, w, q) where p, q are states, $\sigma \in \Sigma$ is a symbol read on the input tape, and $w \in \Sigma^*$ is a word (possibly empty) written on the output tape. A finite transducer execution, i.e., a sequence of successive transitions $\rho = (p_1, \sigma_1, w_1, p_2) \ldots (p_{n-1}, \sigma_n, w_n, p_n)$, operates on the input word $\mathsf{in}(\rho) = \sigma_1 \ldots \sigma_n$ and produces the output word $\mathsf{out}(\rho) = w_1 \ldots w_n$. The semantics of a finite transducer is the set of pairs $(\mathsf{in}(\rho), \mathsf{out}(\rho))$ for all accepting executions ρ. Two different executions ρ_1 and ρ_2 might define the same input/output pair, but can produce the output in a different way because the output words w_i occurring on the transitions of the two runs may differ, although their whole concatenation is the same.

Origin information. Differences in the way transducers produce their output are best captured by the notion of *origin information*, initially proposed for streaming string transducers [10]. An origin function maps any position of the output word to the input position where it was produced. In an execution such as ρ above, any position of w_i has origin i. Figure 1 illustrates five different ways of producing the same output word a^4b^4 (the input word of length 4 is irrelevant here and not depicted). Consider for example the execution ρ_1. When reading the first input symbol (timestep $t = 1$), nothing is produced. At timestep $t = 2$, a^4 is produced, and its positions have origin 2, as depicted to the right of the figure. The sequence b^4 is produced at timestep $t = 4$ so its corresponding positions have origin 4. Note that ρ_3, ρ_4, ρ_5 do not correspond to finite transducer executions, as a^4b^4 is not produced from left to right.

Delay for finite transducers. A natural way of comparing two finite transducer executions ρ_1, ρ_2 on the same input and producing the same output, is to compare the origin functions o_1, o_2 they induce, and in particular how much one is ahead of the other. In Figure 1, both ρ_1 and ρ_2 produce the same output from left to right, yet at different speed: ρ_1 produces bigger chunks of output at lower frequency. The *delay* between both executions is 2 for $t \in \{1, 3\}$ and 0 for $t \in \{2, 4\}$. The global delay $\mathsf{delay}(\rho_1, \rho_2)$ is defined as the maximal delay along the executions, in this case 2. Two transducers T_1 and T_2 are said to be *k-delay equivalent* if for each run ρ_1 of T_1 or T_2 the other transducer has a run ρ_2 with the same input and output satisfying $\mathsf{delay}(\rho_1, \rho_2) \leq k$. This delay notion enjoys two important properties:

Regularity: While the set $E = \{(\rho_1, \rho_2) \mid \mathsf{in}(\rho_1) = \mathsf{in}(\rho_2) \wedge \mathsf{out}(\rho_1) = \mathsf{out}(\rho_2)\}$ is not regular[1] in general for finite transducer executions, its restrictions to pairs of executions with bounded delay is regular: For every $k \in \mathbb{N}$, the set $E_k = \{(\rho_1, \rho_2) \in E \mid \mathsf{delay}(\rho_1, \rho_2) \leq k\}$ is regular [24]. As finite automata compose well with other abstract machines, this allows to use the delay in all kinds of constructions while preserving good decidability properties. For instance k-delay equivalence is decidable [24].

Completeness: For any two finite transducers T_1, T_2 defining string-to-string *functions*, there exists a computable bound k such that T_1 and T_2 are equivalent iff they are k-delay equivalent. The completeness even holds in broader classes [24], such as *finite-valued* transducers, for which there is a global bound on the number of outputs mapped to a single input.

Applications. Due to its regularity and completeness, the notion of delay, while basic, proves to be very powerful, and is omnipresent in the study of finite transducers. It provides decidable approximations of various undecidable decision problems, which reveals to be exact for broad classes of transducers, such as finite-valued transducers [24]. Various patterns characterizing important subclasses of transducers are based on the delay notion: For instance, transducers which are *sequential* [14, 9] (i.e., definable by an input-deterministic transducer); equivalent to finite union of sequential functions [27, 20]; finite-valued [29, 31]. Moreover, it has been used to show that any finite-valued transducer can be decomposed as a union of 1-valued transducers [33, 30]. Canonical notions for finite transducers are also based on delay: for input-deterministic transducers [15] and 1-valued transducers [28].

Streaming string transducers and regular functions. *Streaming string transducers* (SST) are obtained by equipping deterministic finite automata with a finite set of registers that store output [1]. Those registers cannot be tested, but are updated by concatenating them, or prepending/appending some symbols to them. E.g., consider a single-state SST with a loop which, whatever it reads as input, updates its single register O by the operation $O := aOb$, and finally outputs the whole content of O. The execution ρ_5 in Figure 1 represents an execution of this SST. Consider an equivalent single-state SST with two registers X, Y which, whatever the input, executes $X := Xa$ and $Y := bY$, and finally outputs the concatenation XY (ρ_4 in Figure 1 produces the output in this way). While equivalent, those two transducers are not equivalent if the origin information is included in the semantics.

SSTs define a robust set of string-to-string functions, called *regular functions*, which can be equivalently defined by deterministic two-way transducers [1], monadic second-order transductions [16, 22, 4], regular transducer expressions [5, 8, 18, 19], and a logic with origins [17]. Regular functions also enjoy a Krohn-Rhodes decomposition theorem [11]. SSTs have a decidable equivalence problem [26], have been applied to the verification of *list-processing programs* [2], and have been implemented for evaluating string transformations [3].

There are two natural ways of extending the delay defined for finite transducers to compare executions of SSTs with the same input and output: we can either simply compare the number of produced output letters without caring about the positions where it is produced, or we can count the number of positions whose output has already been produced in one run but not in the other. Both methods match the previously defined delay if the output is produced from left to right, but unfortunately neither preserves conjointly the regularity and

[1] Regularity here is defined as classical regularity of word languages, where a pair (ρ_1, ρ_2), are encoded as a single word over a product alphabet of transitions.

the completeness once other output production mechanisms, such as in SSTs, are considered[2]. As the basic notions of delay for SSTs fail to satisfy good properties, better ways of comparing SSTs were introduced, yet none that conjointly has good decidability *and* completeness properties. For instance, *bounded regular resynchronizers* can alter origin functions of SSTs in a controlled manner while preserving good decidability properties, but they are are not complete [13, 12]. In the restricted setting of *single-register* SSTs, a regular and complete notion of delay is defined based on word equations [25]. This approach was applied to prove a decomposition theorem for single-register SSTs, but unfortunately fails when more registers are allowed.

Contributions. We define a delay notion for comparing SST executions, based on the origin functions they induce, that is both regular (Theorem 5) and complete for fundamental SST decision problems including equivalence (Theorem 3 and Corollaries 10 and 11). This delay notion is described in the next subsection. We first present our results. Since our notion is based on origins, it more generally applies to regular functions with origin information. SSTs are known to be equivalent to deterministic two-way and MSO transducers, via origin-preserving encodings [10], so we obtain as a corollary that our delay notion is also complete for equivalence of deterministic two-way transducers and MSO transducers (Corollary 4).

The origin semantics allows us to recover decidability of several decision problems: While non-deterministic SST equivalence is undecidable, we can decide whether two non-deterministic SSTs define the same relations *with same origins*. However, asking for identical origins is very restrictive: it fails to detect equivalence if the origins are perturbed ever so slightly. This observation was already made in [24] in the context of finite transducers, where it is proposed to relax several decision problems, such as equivalence with same origins, to decision problems with *close* origins, such as equivalence up to a given delay bound k. We prove that the inclusion, equivalence and variable minimization problems up to a given delay bound are decidable for (non-deterministic) SSTs (Theorems 8 and 9), while in general those problems are undecidable. Since our delay notion is complete for equivalence of (deterministic) SSTs, the latter result provides an alternative proof that such SSTs have decidable equivalence problem, and sheds light on the intrinsic reasons why SSTs executions are equivalent.

Delay for SSTs. Our notion of delay is based on the following observation: The order of production of each output segment that is a power of a small word should have no impact on the delay. For every $\ell \in \mathbb{N}$, we introduce the measure delay_ℓ, which first decomposes the output into blocks that are powers of words smaller than or equal to ℓ. Then, at any position j ending a block (instead of *any* position), we measure the maximal difference at any timestep t, between the number of output positions at the left of j produced before timestep t in the first and second executions. We then take the maximal such value for all j. Consider ρ_4 and ρ_5 for example. delay_1 splits the output a^4b^4 of Figure 1 into two blocks, a^4 and b^4. Consider ending block position $j = 4$. At any timestep, the maximal difference of quantity of outputs produced at the left of j is always 0, because ρ_4, ρ_5 produce exactly one symbol per timestep before position $j = 4$. The same reasoning applies for ending block position $j = 8$, where here instead, exactly two symbols are produced by ρ_4, ρ_5 at any timestep, at the left of position $j = 8$. Hence $\mathsf{delay}_1(\rho_4, \rho_5) = 0$. We refer to Section 3 for a smooth introduction

[2] We refer the interested reader to the appendix for more details about those natural extensions and an explanation of why they are not satisfactory.

to our delay notion and the main results and to Section 4.1 for a proof of its completeness, and to Section 4.2 for a proof of its regularity. The completeness proof is perhaps the most involved. It is based on a key pumping lemma (Lemma 6), which intuitively states that for any SST, there exist computable bounds k and ℓ such that, if the delay delay_ℓ between two executions on the same input/output is greater than k, then those two executions can be pumped to construct two executions over the same input but with *different* outputs.

2 Preliminaries

We fix our notation. Let $\mathbb{N}$ denote the set of non-negative integers. The interval between integers a and b including resp. excluding a and b is denoted $[a, b]$ resp. (a, b).

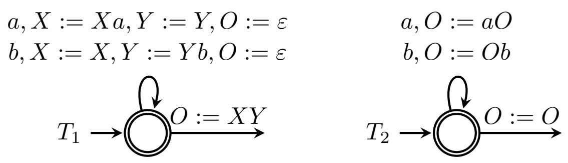
Free monoid. Given a finite alphabet Σ, the *free monoid* over Σ is the monoid $(\Sigma^*, \cdot, \varepsilon)$ defined as follows. The set Σ^* is composed of finite sequences of elements of Σ, called *words*. The operation $\cdot$ is the usual word concatenation. The neutral element ε is the empty word.

A subset $L \subseteq \Sigma^*$ is called a *language*. Given a word $u = a_1 \cdots a_n \in \Sigma^*$, $|u|$ denotes its length, $u[i]$ denotes its ith letter a_i, $u[i, j]$ denotes its infix $a_i \cdots a_j$. Given an additional word $v = b_1 \cdots b_n \in \Sigma^*$, $u \otimes v$ denote the *convolution* $\binom{a_1 \cdots a_n}{b_1 \cdots b_n} \in (\Sigma \times \Sigma)^*$.

Automaton. A *(finite state) automaton* is a tuple $A = (\Sigma, Q, I, \Delta, F)$ composed of a finite alphabet Σ, a finite set of states Q, a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, and a set of transitions $\Delta \subseteq Q \times \Sigma \times Q$. A *run* of A is a sequence $\rho = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n \in Q(\Sigma Q)^*$ such that $(q_{i-1}, a_i, q_i) \in \Delta$ for every $1 \leq i \leq n$. The *input* of ρ is the word $a_1 a_2 \cdots a_n \in \Sigma^*$. The run ρ is called *accepting* if its first state is initial ($q_0 \in I$) and its last state is final ($q_n \in F$). The automaton A is *deterministic* if I is a singleton and Δ is expressed as a function $\delta : Q \times \Sigma \to Q$. The *language recognized* by A is the set $L(A) \subseteq \Sigma^*$ composed of the inputs of all its accepting runs.

Substitutions monoid. Given a finite alphabet Σ, a finite set $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ of *variables*, and a designated *output variable* $O \in \mathcal{X}$, the *(copyless) substitutions monoid* $(\mathcal{S}_{\mathcal{X},\Sigma}, \circ, \mathsf{Id}_{\mathcal{X}})$ is defined as follows. The set $\mathcal{S}_{\mathcal{X},\Sigma}$ contains all functions $\sigma : \mathcal{X} \to (\mathcal{X} \cup \Sigma)^*$ such that no variable appears twice in the image of a variable and no variable appears in the images of two distinct variables, i.e., the word $\sigma(X_1) \ldots \sigma(X_n)$ does not contain twice the same variable. The composition $\sigma_1 \circ \sigma_2$ of two substitutions $\sigma_1, \sigma_2 \in \mathcal{S}_{\mathcal{X},\Sigma}$ is obtained by first applying σ_2, and then σ_1. Formally, it is the function $\hat{\sigma}_1(\hat{\sigma}_2(\cdot))$, where each substitution σ is morphically extended to $\hat{\sigma}$ over $(\mathcal{X} \cup \Sigma)^*$ by letting $\hat{\sigma}(X) = \sigma(X)$ and $\hat{\sigma}(\alpha) = \alpha$ for $\alpha \in \Sigma$. We write σ in place of $\hat{\sigma}$. For instance, the substitution composition $(\sigma_1 \colon X \mapsto \varepsilon) \circ (\sigma_2 \colon X \mapsto aX) \circ (\sigma_3 \colon X \mapsto bXc)$ maps X to bac, since $\sigma_3(X) = bXc$, $\sigma_2(bXc) = baXc$ and $\sigma_1(baXc) = bac$. The neutral element $\mathsf{Id}_{\mathcal{X}}$ is the identity function mapping each variable $X \in \mathcal{X}$ to itself. Finally, we denote by σ_ε the substitution mapping each variable $X \in \mathcal{X}$ to ε. We recall that O is the designated output variable; the *output* of a substitution σ is the word $\mathsf{out}(\sigma) = (\sigma_\varepsilon \circ \sigma)(O)$.

Streaming string transducer. A *streaming string transducer* (SST) is a tuple $T = (\Sigma, Q, I, \delta, F, \mathcal{X}, O, \kappa, \kappa_F)$ where $A_T = (\Sigma, Q, I, \delta, F)$ is a deterministic automaton, called *underlying automaton* of T, $\mathcal{X}$ is a finite set of variables (also called registers), $O \in \mathcal{X}$ is a final variable, $\kappa \colon \delta \to \mathcal{S}_{\mathcal{X},\Sigma}$ is an output function, and $\kappa_F \colon Q \to \mathcal{S}_{\mathcal{X},\Sigma}$ is a final output function. A *run* of T is a run $\rho = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n \in Q(\Sigma Q)^*$ of A_T, we define $\kappa(\rho)$ as the substitution



$a, X := Xa, Y := Y, O := \varepsilon$
$b, X := X, Y := Yb, O := \varepsilon$
T_1
$O := XY$
$a, O := aO$
$b, O := Ob$
T_2
$O := O$

Figure 2 Two deterministic SSTs that realize the same sorting function, cf. Example 1.

obtained by composing sequentially all substitutions occurring on the transitions and the final substitutions, i.e. $\kappa(\rho) = \kappa((q_0, a_1, q_1)) \circ \kappa((q_1, a_2, q_2)) \circ \cdots \circ \kappa((q_{n-1}, a_n, q_n)) \circ \kappa_F(q_n) \in \mathcal{S}_{\mathcal{X},\Sigma}$. The *output* of the run ρ is the word $(\sigma_\varepsilon \circ \kappa(\rho))(O)$. The *transduction* $R(T)$ *recognized* (also called *realized*) by T is the set of pairs $(u, v) \in \Sigma^* \times \Sigma^*$ such that T has an accepting run with input u and output v. Non-deterministic SST are defined the same way except that A_T is non-deterministic. We emphasize that the transduction realized by a (deterministic) SST is a partial function.

▶ **Example 1.** Depicted in Figure 2 are deterministic SSTs T_1 and T_2 that realize the same sorting function $f\colon \{a,b\}^* \to \{a,b\}^*$ that maps $u \in \{a,b\}^*$ to $a^m b^n$, where m (resp. n) is the number of occurrences of a (resp. b) in u.

Origin information. A *word with origins* is a word $\tilde{u} := u \otimes o \in (\Sigma \times \mathbb{N})^*$ where each letter of u is annotated with a natural number signifying its origin in time. A *transduction with origins* is a relation $R \subseteq \Sigma^* \times (\Sigma \times \mathbb{N})^*$.

Naturally, we associate with a sequence $\lambda = \sigma_1\sigma_2 \cdots \sigma_n \in \mathcal{S}^*_{\mathcal{X},\Sigma}$ of substitutions the output word with origins $\tilde{\mathsf{out}}(\lambda) = \mathsf{out}(\lambda) \otimes i_1 \cdots i_{|\mathsf{out}(\lambda)|}$ with $i_j = \mathsf{out}(\sigma'_1\sigma'_2 \cdots \sigma'_n)[j]$ for $1 \leq j \leq |\mathsf{out}(\lambda)|$ where for all $1 \leq t \leq n$, the substitution $\sigma'_t \in \mathcal{S}_{\mathcal{X},\mathbb{N}}$ is obtained by replacing each output letter in σ_t with the number t. Furthermore, we associate with an SST T the *transduction with origins* $R_{\mathsf{ori}}(T)$ that is the set of pairs $(u, \tilde{\mathsf{out}}(\lambda)) \in \Sigma^* \times (\Sigma \times \mathbb{N})^*$ such that T has an accepting run ρ with input u and sequence of substitutions $\kappa(\rho) = \lambda$. For example, regarding the SSTs T_1 and T_2 from Figure 2, we obtain that $(abaa, \binom{aaab}{1342}) \in R_{\mathsf{ori}}(T_1)$ and $(abaa, \binom{aaab}{4312}) \in R_{\mathsf{ori}}(T_2)$.

3 Delay measure

As mentioned in the introduction, the concept of delay has proven to be a useful tool in the understanding of finite transducers. Our goal is to introduce a robust delay measure suitable to gain a better understanding of streaming string transducers. Towards that, we informally recall the notion of delay for finite transducers: The delay between two finite transducer computations that produce the same output in the end, is a measure for how much ahead one output is compared to the other during the computation. In other words, the difference between the length of the so-far produced output is measured.

A key difference between finite transducers and SSTs is that the former build their outputs from left to right while SSTs do not have this restriction. To design a notion of delay taking this into account, we use origin information to define a notion of *weight difference*. These notions are illustrated in Example 2.

Weight difference. Given a word with origins $\tilde{u} := u \otimes o \in (\Sigma \times \mathbb{N})^*$, a time $t \in \mathbb{N}$, and $j \in \mathbb{N}$, we define the *positional weight* $\mathsf{weight}_{j,t}(\tilde{u}) \in \mathbb{N}$ as the number of positions of u up to j whose origin is no later than t, i.e.,

$$\mathsf{weight}_{j,t}(\tilde{u}) = \big|\{i \in \{1, 2, \ldots, \min(j, |u|)\} \mid o[i] \leq t\}\big|.$$

Note that $\mathsf{weight}_{j,t}(\tilde{u}) = \mathsf{weight}_{|u|,t}(\tilde{u})$ for all $j \geq |u|$. Given a second word with origins $\tilde{v} \in (\Sigma \times \mathbb{N})^*$, and $j_1, j_2 \in \mathbb{N}$, we define the *weight difference* as

$$\mathsf{diff}_{j_1,j_2,t}(\tilde{u},\tilde{v}) = \left|\mathsf{weight}_{j_1,t}(\tilde{u}) - \mathsf{weight}_{j_2,t}(\tilde{v})\right| \text{ and } \mathsf{diff}_{j,t}(\tilde{u},\tilde{v}) = \mathsf{diff}_{j,j,t}(\tilde{u},\tilde{v}).$$

Furthermore, we define the *maximal weight difference* as

$$\mathsf{max\text{-}diff}_{j_1,j_2}(\tilde{u},\tilde{v}) = \max_{t\in\mathbb{N}} \left(\mathsf{diff}_{j_1,j_2,t}(\tilde{u},\tilde{v})\right) \text{ and } \mathsf{max\text{-}diff}_{j}(\tilde{u},\tilde{v}) = \max_{t\in\mathbb{N}} \left(\mathsf{diff}_{j,t}(\tilde{u},\tilde{v})\right).$$

We remark that the value of $\mathsf{max\text{-}diff}$ is bounded even though t takes infinitely many values. Also, we note that $\mathsf{max\text{-}diff}_0(\tilde{u},\tilde{v}) = 0$. We illustrate these notions.

▶ **Example 2.** We base our example on the SSTs T_1, T_2 from Figure 2. On input *abaaa* both SSTs produce output *aaaab*. The associated origins differ, we have $\tilde{u} = \binom{aaaab}{13452}$ and $\tilde{v} = \binom{aaaab}{54312}$. We obtain $\mathsf{diff}_{2,0}(\tilde{u},\tilde{v}) = 0$, $\mathsf{diff}_{2,1}(\tilde{u},\tilde{v}) = 1$, $\mathsf{diff}_{2,2}(\tilde{u},\tilde{v}) = 1$, $\mathsf{diff}_{2,3}(\tilde{u},\tilde{v}) = 2$, $\mathsf{diff}_{2,4}(\tilde{u},\tilde{v}) = 1$, $\mathsf{diff}_{2,5}(\tilde{u},\tilde{v}) = 0$, and $\mathsf{max\text{-}diff}_2(\tilde{u},\tilde{v}) = 2$. More generally, on input aba^i the output is $a^{i+1}b$ and with origins we have $\tilde{u}_i = \left(\begin{smallmatrix} a & a & a & \cdots & a & a & b \\ 1 & 3 & 4 & \cdots & i+1 & i+2 & 2 \end{smallmatrix}\right)$ and $\tilde{v}_i = \left(\begin{smallmatrix} a & a & \cdots & a & a & a & b \\ i+2 & i+1 & \cdots & 4 & 3 & 1 & 2 \end{smallmatrix}\right)$ for all $i > 0$. Moreover, $\mathsf{max\text{-}diff}_{(i+1)/2}(\tilde{u}_i,\tilde{v}_i) = (i+1)/2$ for all odd i.

The first idea of a delay notion for SSTs similar to finite transducers is to consider the maximal weight difference that can occur. In Example 2, since T_1 builds the a-output block from left to right and T_2 from right to left, their maximal weight difference is unbounded even though $T_1 \equiv T_2$. Hence, this first idea of a delay notion violates the completeness requirement. To avoid this problem, we would like our notion of delay to reflect that, for a periodic block, it is not important if a repetition of the period is appended or prepended: the result is the same. Therefore, we only measure the difference at the end of periodic blocks and not inside of them. To this end, we introduce a new notion.

Factors, cuts. The *primitive root* of a word $u \in \Sigma^*$, denoted $\mathsf{root}(u)$, is the shortest word w such that $u = w^k$ for some positive integer k. We call a word $u \in \Sigma^*$ *primitive* if $\mathsf{root}(u) = u$.

Let $u \in \Sigma^*$ and $\ell > 0$. We cut u into factors such that each factor has a primitive root of length at most ℓ. The factors are chosen inductively from left to right in a way to maximize the size of each factor as follows. The first factor u_1 of u is the longest prefix of u such that $|\mathsf{root}(u_1)| \leq \ell$, the ith factor u_i of u is the longest prefix of u' such that $|\mathsf{root}(u_i)| \leq \ell$ where $u = u_1 \cdots u_{i-1}u'$. We refer to this unique *ℓ-factorization* as $\mathsf{factors}_\ell(u)$. Moreover, we denote by $\mathsf{cut}_\ell(u)$ the set that contains the end positions of the factors referred to as *ℓ-cuts*. For example, consider $u = aaababcbabaaaaa$ and $\ell = 2$, then the unique ℓ-factorization is $aaa|ba|bc|baba|aaaa|$ and its ℓ-cuts are $\{3, 5, 7, 11, 15\}$.

Delay. We define delay for a word and two origin annotations (of said word) by considering the maximal weight difference *at the cut positions* which are obtained via the factorization into periodic words as introduced above. Formally, given a word $w \in \Sigma^*$ and two annotated versions $\tilde{u}, \tilde{v}$ with origin, i.e., $\tilde{u} := w \otimes o_1, \tilde{v} := w \otimes o_2 \in (\Sigma \times \mathbb{N})^*$, we define the *$\ell$-delay* $\mathsf{delay}_\ell(\tilde{u},\tilde{v})$ (delay for short) as

$$\mathsf{delay}_\ell(\tilde{u},\tilde{v}) = \max_{j\in\mathsf{cut}_\ell(w)} \mathsf{max\text{-}diff}_j(\tilde{u},\tilde{v}).$$

Going back to Example 2, we have $\mathsf{delay}_1(\tilde{u}_i,\tilde{v}_i) = 0$, because the 1-factorization of $a^{i+1}b$ is $a^{i+1}|b|$ and we have $\mathsf{max\text{-}diff}_{i+1}(\tilde{u}_i,\tilde{v}_i) = \mathsf{max\text{-}diff}_{i+2}(\tilde{u}_i,\tilde{v}_i) = 0$.

We apply the delay notion to transductions with origin. Intuitively, two such transductions are "close" if for every pair $(u, w \otimes o_1)$ (from one transduction) there is some pair $(u, w \otimes o_2)$ (from the other transduction) such that the delay between these outputs with origins is small.

Let R_1, R_2 denote transductions with origin. We say that R_1 is (k,ℓ)-*included* in R_2 (written $R_1 \subseteq_{k,\ell} R_2$) if the following holds: for all $(u, w \otimes o_1) \in R_1$, there exists $(u, w \otimes o_2) \in R_2$ such that $\mathsf{delay}_\ell(w \otimes o_1, w \otimes o_2) \leq k$. We say that R_1 is (k,ℓ)-*equivalent* to R_2 (written $R_1 \equiv_{k,\ell} R_2$) if $R_1 \subseteq_{k,\ell} R_2$ and conversely $R_2 \subseteq_{k,\ell} R_1$. We are ready to state our first main result which illustrates the generality of our delay notion.

▶ **Theorem 3** (Completeness). *Given two SSTs T_1 and T_2, there exist computable integers k, ℓ such that $T_1 \equiv T_2$ iff $R_{\mathsf{ori}}(T_1) \equiv_{k,\ell} R_{\mathsf{ori}}(T_2)$.*

Section 4.1 is devoted to the proof of Theorem 3 and Section 5 illustrates some consequences of Theorem 3. As the delay between streaming string transducers only depends on their induced transductions with origin, we are able to state a more general result. Corollary 4 uses the fact that deterministic streaming string transducers, deterministic two-way transducers and MSO transducers are equally expressive – they characterize the so-called *regular functions* – and every regular function given in one formalism can be translated into every other one without changing its induced transduction with origins [16, 22, 4].

▶ **Corollary 4.** *Given deterministic two-way transducers resp. MSO transducers T_1 and T_2. Let R_1 and R_2 denote their induced transductions with origin. There exist computable integers k, ℓ such that $T_1 \equiv T_2$ iff $R_1 \equiv_{k,\ell} R_2$.*

We focus on the second main aspect of our delay measure, namely, regularity. Meaning that for every $k, \ell \in \mathbb{N}$, we would like to construct a finite automaton that accepts (suitable representations of) pairs $(w \otimes o_1, w \otimes o_2)$ with $\mathsf{delay}_\ell(w \otimes o_1, w \otimes o_2) \leq k$. As finite automata enjoy good closure properties, this yields a useful tool to solve for instance decision problems up to fixed delay, cf. Section 5. As mentioned in the paragraph before Corollary 4, our delay measure is applicable to transductions with origins and is complete for several transducer models. However, we need to pick some way to represent such transductions to show regularity. Hence, we prove our second main result for streaming string transducers.

▶ **Theorem 5** (Regularity). *Let $\mathcal{S} \subseteq \mathcal{S}_{\mathcal{X},\Sigma}$ be a* finite *subset of $\mathcal{S}_{\mathcal{X},\Sigma}$, let $k \geq 0$ and $\ell > 0$. The following set is a regular language:*

$$\mathbb{D}_{k,\ell,\mathcal{S}} = \{\lambda \otimes \mu \in (\mathcal{S} \times \mathcal{S})^* \mid \mathit{delay}_\ell(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k \text{ and } |\lambda| = |\mu|\}.$$

Note that $\lambda \otimes \mu \in \mathbb{D}_{k,\ell,\mathcal{S}}$ implies $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$ because delay_ℓ is defined only for such substitution sequences. We write $\mathbb{D}_{k,\ell}$ instead of $\mathbb{D}_{k,\ell,\mathcal{S}}$ when $\mathcal{S}$ is clear from the context. We prove Theorem 5 in Section 4.2 and show some applications of this result in Section 5.

4 Completeness and regularity

4.1 Completeness of the delay notion

We now prove the completeness result for SSTs, as stated in Theorem 3. To this end, we show that whenever the delay between two sequences of substitutions is sufficiently large we can pump well-chosen factors to obtain two sequences of substitutions producing outputs that are distinct. Formally, given a sequence of substitutions $\lambda \in \mathcal{S}^*$ and $1 \leq s < t < |\lambda|$, we denote by $\mathsf{pump}_{(s,t]}(\lambda)$ the sequence of substitutions $\lambda[1,t]\lambda(s,t]\lambda(t,|\lambda|)$ obtained by pumping the interval $(s,t]$, and we prove the following:

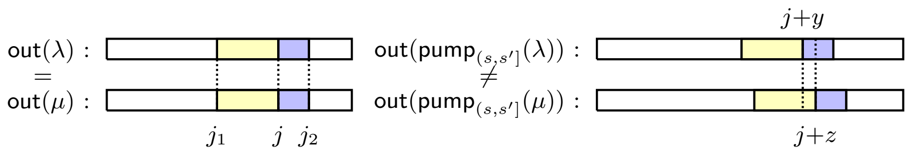


Figure 3 Illustration of the main idea used in the proof of Lemma 6.

▶ **Lemma 6.** *Let $\mathcal{S}$ be a finite set of substitutions. There exist computable integers $k, \ell \in \mathbb{N}$ such that for every integer $C \in \mathbb{N}$ and every pair $\lambda, \mu \in \mathcal{S}^*$ that satisfy $|\lambda| = |\mu|$, $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$, and $\mathsf{delay}_{C\ell}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > C^2k$, there exist $0 \le t_1 < t_2 < \ldots < t_C < |\lambda|$ satisfying*

$$\mathsf{out}(\mathsf{pump}_{(t_i,t_j]}(\lambda)) \ne \mathsf{out}(\mathsf{pump}_{(t_i,t_j]}(\mu)) \text{ for every } 1 \le i < j \le C.$$

Moreover we can choose k and ℓ exponential with respect to the number of variables of $\mathcal{S}$. Before delving into the proof of Lemma 6, we argue that Theorem 3 follows as a corollary.

Proof of Theorem 3. Let T_1 and T_2 be two SSTs with set of states Q_1 and Q_2. By symmetry, it is sufficient to show that $R(T_1) \subseteq R(T_2)$ iff $R_{\mathsf{ori}}(T_1) \subseteq_{C^2k,C\ell} R_{\mathsf{ori}}(T_2)$, where $C = |Q_1| \cdot |Q_2| + 1$, and k, ℓ are as in the statement of Lemma 6 with respect to the union $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ of the substitutions used by T_1 and T_2. The right to left direction of the 'iff' is immediate, as inclusion up to bounded delay is stronger than inclusion.

We now apply Lemma 6 to prove the converse direction. Suppose that $R_{\mathsf{ori}}(T_1) \not\subseteq_{C^2k,C\ell} R_{\mathsf{ori}}(T_2)$. Then there exists a pair with origins $(u, w \otimes o_1) \in R_{\mathsf{ori}}(T_1)$ such that all the pairs with origin $(u, w \otimes o_2) \in R_{\mathsf{ori}}(T_2)$ with matching input and output satisfy $\mathsf{delay}_{C\ell}(w \otimes o_1, w \otimes o_2) > C^2k$. Two possible cases arise: either there is no pair of the form $(u, w \otimes o_2)$ in $R_{\mathsf{ori}}(T_2)$, or there exists such a pair $(u, w \otimes o_2) \in R_{\mathsf{ori}}(T_2)$, and it satisfies $\mathsf{delay}_{C\ell}(w \otimes o_1, w \otimes o_2) > C^2k$. In the former case, we immediately get that $R(T_1) \not\subseteq R(T_2)$, since (u, w) is in $R(T_1)$ but not in $R(T_2)$. In the latter case, we get that there exists a run ρ_1 of T_1 and a run ρ_2 of T_2 over the same input u that both produce the same output w, but with very different origins functions: $\mathsf{delay}_{C\ell}(\tilde{\mathsf{out}}(\kappa_1(\rho_1)), \tilde{\mathsf{out}}(\kappa_2(\rho_2))) > C^2k$. Then Lemma 6 implies that we can find C intermediate points in $\kappa_1(\rho_1)$ and $\kappa_2(\rho_2)$ such that iterating the segment between any two points yields sequences of substitutions producing distinct outputs. As $C = |Q_1| \cdot |Q_2| + 1$, two of these points mark a loop in both ρ_1 and ρ_2, and pumping these loops creates runs of T_1 and T_2 with the same input but different outputs. Since T_2 is deterministic, and therefore cannot map an input word to two distinct output words, this implies that $R(T_1)$ is not included in $R(T_2)$. ◀

Proof overview of Lemma 6. We consider two substitution sequences $\lambda, \mu \in \mathcal{S}^*$ that have the same length, produce the same output, and satisfy $\mathsf{delay}_{C\ell}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > C^2k$ for some $C, k, \ell \in \mathbb{N}$. This implies the existence of a cut position $j \in \mathsf{cut}_{C\ell}(\mathsf{out}(\lambda))$ and a point $t \in [0, |\lambda|)$ for which the weight difference $\mathsf{diff}_{j,t}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu))$ is equal to C^2k.

Our proof is based on the following fact, illustrated by Figure 3: If we choose k such that $\mathsf{diff}_{j,t}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu))$ is sufficiently large, there are intervals $(s, s'] \subset [1, |\lambda|)$ that, once pumped, add distinct amount of output before position j, thus creating a misalignment between two copies of the letter $\mathsf{out}(\lambda)[j]$ in the output words generated by $\mathsf{pump}_{(s,s']}(\lambda)$ and $\mathsf{pump}_{(s,s']}(\mu)$. Moreover, we show that carefully identifying patterns occurring along the two substitution sequences also allows us to ensure that some neighborhood $\mathsf{out}(\lambda)[j_1, j_2] = \mathsf{out}(\mu)[j_1, j_2]$ of the position j is preserved in both $\mathsf{out}(\mathsf{pump}_{(s,s']}(\lambda))$ and $\mathsf{out}(\mathsf{pump}_{(s,s']}(\mu))$, in a way that these two copies overlap, but not perfectly (this is the most complex part of the proof). At this point, we use the fact that j is a cut position of $\mathsf{out}(\lambda)$: since j marks the position where the period changes, and this position is not aligned properly in $\mathsf{out}(\mathsf{pump}_{(s,s']}(\lambda))$ and $\mathsf{out}(\mathsf{pump}_{(s,s']}(\mu))$, we can derive the existence of a mismatch, proving that $\mathsf{out}(\mathsf{pump}_{(s,s']}(\lambda)) \neq \mathsf{out}(\mathsf{pump}_{(s,s']}(\mu))$.

Then, all that remains to do is to combine a few counting arguments to show how to choose k and ℓ sufficiently large with respect to the parameters of $\mathcal{S}$ (and, crucially, independently of C) so that the fact that $\mathsf{diff}_{j,t}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > C^2k$ implies the existence of C consecutive intervals $(s,s'] \subset [1,|\lambda|)$, which, as described earlier, satisfy $\mathsf{out}(\mathsf{pump}_{(s,s']}(\lambda)) \neq \mathsf{out}(\mathsf{pump}_{(s,s']}(\mu))$, which concludes the proof of Lemma 6. ◀

4.2 Regularity of the delay notion

Our goal is to prove that the delay notion is regular, as stated in Theorem 5. All over this section, k and ℓ are non-negative integers, and $\mathcal{S}$ is a finite set of substitutions over a finite set of variables $\mathcal{X}$ and an alphabet Σ. Lemma 7 characterizes the pairs of substitution sequences whose outputs end with a unique endmarker symbol $\dashv$ and that are *not* in $\mathbb{D}_{k,\ell,\mathcal{S}}$, using properties which are independently shown to be regular. We first start with the characterization, then give an overview on how to show its regularity.

A characterization. Note that a pair (λ, μ) of two substitution sequences of the same length is not in $\mathbb{D}_{k,\ell,\mathcal{S}}$ if either $\mathsf{out}(\lambda) \neq \mathsf{out}(\mu)$ or there is a cut witnessing a delay greater than k. Unfortunately, the first condition $\mathsf{out}(\lambda) \neq \mathsf{out}(\mu)$ is not regular. So in order to characterize the complement of $\mathbb{D}_{k,\ell,\mathcal{S}}$ by regular properties, we somehow have to mix conditions on differences in the output and on positions witnessing a big delay. For the corresponding formal statement in Lemma 7 we need the following definition.

Given a sequence of substitutions λ and $i \geq 0$, let $\mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ be the smallest output position j such that $j > i$ and $j \in \mathsf{cut}_\ell(\mathsf{out}(\lambda))$, if it exists. Note that, as the last output position is a cut, such a position j always exists unless i is the last output position. Formally, $\mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ denotes the set $\min(\mathsf{cut}_\ell(\mathsf{out}(\lambda)) \cap \{j \mid j > i\})$. The result is either a singleton or the empty set. In the former case, we write $\mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda)) = j$ instead of $\mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda)) = \{j\}$.

▶ **Lemma 7.** *Let $\dashv \in \Sigma$ and λ, $\mu \in \mathcal{S}^*$ be sequences with $\mathsf{out}(\lambda), \mathsf{out}(\mu) \in (\Sigma \setminus \{\dashv\})^*\dashv$ and $|\lambda| = |\mu|$. Then $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$ iff there exists $i \in (\mathsf{cut}_\ell(\mathsf{out}(\lambda)) \cap \mathsf{cut}_\ell(\mathsf{out}(\mu))) \cup \{0\}$ such that $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$, and one of the following holds:*

1. *Both $j_1 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ and $j_2 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\mu))$ exist, and either $j_1 \neq j_2$ or $\mathsf{max\text{-}diff}_{j_1,j_2}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$;*
2. *$\mathsf{out}(\lambda)[i+b] \neq \mathsf{out}(\mu)[i+b]$ for some $b \in [0, \ell^2]$.*

Proof sketch. Assume that λ, μ satisfy Item 1 or 2 from the above statement. If $\mathsf{out}(\lambda) \neq \mathsf{out}(\mu)$, then clearly $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$. So assume that $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$. Then $\mathsf{cut}_\ell(\mathsf{out}(\lambda)) = \mathsf{cut}_\ell(\mathsf{out}(\mu))$. Hence Item 1 is satisfied with $j_1 = j_2 =: j$ and $\mathsf{max\text{-}diff}_j(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$. Since j is a cut, we obtain that $\mathsf{delay}_\ell(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$ and thus $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$.

Conversely, let $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$. First assume $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$. Since $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$, there exists some $j \in \mathsf{cut}_\ell(\mathsf{out}(\lambda)) \cap \mathsf{cut}_\ell(\mathsf{out}(\mu))$ such that $\mathsf{max\text{-}diff}_j(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$, and we can satisfy Item 1. Second, we assume $\mathsf{out}(\lambda) \neq \mathsf{out}(\mu)$. Let m be the position of the earliest mismatch (that is, $\mathsf{out}(\lambda)[m] \neq \mathsf{out}(\mu)[m]$), and i be the nearest common cut to the left of the mismatch. If $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$, we have a common cut with a too large difference before a mismatch occurs. We can treat this situation as if $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$ and satisfy Item 1 as before. If $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$, we show that either the mismatch is close to i (that is, $m \leq i + \ell^2$) and Item 2 is satisfied, or the mismatch causes $j_1 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ and $j_2 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\mu))$ to be different and Item 1 is satisfied for $j_1 \neq j_2$. ◀

Proof overview for the regularity of the delay notion. Let us denote by $\mathbb{C}^{\dashv}_{k,\ell,\mathcal{S}}$ the set of words of the form $\lambda \otimes \mu$ such that $\lambda, \mu \in \mathcal{S}^*$ satisfy the properties of the characterization given in Lemma 7. The main technical part is to show that $\mathbb{C}^{\dashv}_{k,\ell,\mathcal{S}}$ is regular. Then regularity of $\mathbb{D}_{k,\ell,\mathcal{S}}$ follows by complementation and end-marker removal, which preserve regularity.

First, note that the definition of $\mathbb{C}^{\dashv}_{k,\ell,\mathcal{S}}$ is existential in nature: it asks for the existence of positions in $\mathsf{out}(\lambda)$ and $\mathsf{out}(\mu)$ satisfying some properties. A classical way of dealing with positions quantified existentially in automata theory is to mark some positions in the input by using an extended alphabet, construct an automaton over the extended alphabet, and then project this automaton over the original alphabet. Here, the positions needed in $\mathbb{C}^{\dashv}_{k,\ell,\mathcal{S}}$ are positions of $\mathsf{out}(\lambda)$ and $\mathsf{out}(\mu)$, while the automata we want to construct read λ and μ as input. So, instead of marking input positions, we rather mark positions in right-hand sides of updates occurring in the substitutions of λ and μ. Let us make this more precise. First, for $n \geq 0$, words u over the alphabet Σ are extended into *n-marked words*, i.e., words over the alphabet $\Sigma \times 2^{\{1,\dots,n\}}$, such that the additional information in $\{1, \dots, n\}$ precisely corresponds to an n-tuple of positions $\overline{x}$ of u (position x_i is marked with label i for all $i \in \{1, \dots, n\}$, and we consider sets because the same position can correspond to different components of $\overline{x}$). By extension, we also define an operation $\triangleright$ which marks any substitution sequence $\lambda \in \mathcal{S}^*_{\mathcal{X},\Sigma}$ by a tuple $\overline{x}$ of positions of $\mathsf{out}(\lambda)$, resulting in a substitution sequence $(\lambda \triangleright \overline{x}) \in \mathcal{S}^*_{\mathcal{X},\Sigma\times 2^{\{1,\dots,n\}}}$ such that $\mathsf{out}(\lambda \triangleright \overline{x}) = \mathsf{out}(\lambda) \triangleright \overline{x}$.

We show that the set of substitution sequences $\lambda \triangleright i$ satisfying $i \in \mathsf{cut}_\ell(\mathsf{out}(\lambda))$ is regular, and similarly for $\mathsf{next\text{-}cut}_\ell$. To do so, we prove that the set $\{u \triangleright i \mid i \in \mathsf{cut}_\ell(u)\}$ is regular (and similarly for $\mathsf{next\text{-}cut}_\ell$) and then transfer this result to marked substitution sequences, as regular languages are preserved under inverse of SSTs.

Then, we show regularity results for predicates of the form $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$ and $\mathsf{max\text{-}diff}_{j_1,j_2}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$. In the end, all parts of the property of the characterization of Lemma 7 are shown to be regular, so that the whole property can be checked by a synchronized product of automata. Perhaps the most interesting part is how to show that the predicate $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$ is regular. More precisely, it is shown that the set of $(\lambda \triangleright i_1) \otimes (\mu \triangleright i_2)$ such that $i_1 = i_2 = i$ and $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu))$ is smaller than k (which is a given constant), is regular. Let us intuitively explain why. In general, checking whether two marked positions i_1 (in $\mathsf{out}(\lambda)$) and i_2 (in $\mathsf{out}(\mu)$) are equal cannot be done in a regular way (recall that the automaton reads $\lambda \otimes \mu$ and not their outputs). However, if additionally, one has to check that $\mathsf{max\text{-}diff}_{i_1,i_2}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu))$ is smaller than k, it is actually regular. To do so, a finite automaton needs to monitor the difference in the outputs produced in λ and μ before positions i_1 and i_2 resp., and check that it is bounded by k (otherwise it rejects). The difference must eventually reach 0 when the whole inputs λ and μ have been read, to ensure $i_1 = i_2$.

We also prove that once a position i in $\mathsf{out}(\lambda)$ is marked, then the next cut $j_1 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ can be identified in a regular way by an automaton. Similarly, given a constant d, the output position $i + d$ can also be identified in a regular way. This allows us to check the properties of Item 1 and Item 2 respectively of Lemma 7. This concludes the overview of the proof of Theorem 5. A complexity analysis yields that the set $\mathbb{D}_{k,\ell,\mathcal{S}}$ is recognizable by a DFA with a number of states doubly exponential in ℓ^3 and in $|\mathcal{X}|$, and singly exponential in k.

5 Applications of delay completeness and regularity

Decision problems up to fixed delay. Instead of comparing the transductions defined by non-deterministic SSTs for inclusion or equivalence, which are undecidable problems already for finite (variable-free) transducers, we show here that the strengthening of those problems up to fixed delay, $\subseteq_{k,l}$ and $\equiv_{k,l}$, are decidable. The key to decide inclusion and equivalence up to fixed delay is the regularity of the delay notion as stated in Theorem 5. Hence, those problems can be reduced to classical inclusion and equivalence problems of regular languages.

▶ **Theorem 8.** *Given integers k, ℓ, the (k, ℓ)-inclusion problem for non-deterministic SSTs is decidable. It is* PSPACE*-complete if k, ℓ are constants and the number of variables $|\mathcal{X}|$ is a constant. The same results hold for the (k, ℓ)-equivalence problem.*

Proof sketch. We sketch the result for the inclusion problem. For a non-deterministic SST T over input alphabet Σ, we denote by $L(T)$ its language, defined as the set of words of the form $u \otimes \tau(\rho)$, where $u \in \Sigma^*$, ρ is an accepting run of T over u, and $\tau(\rho)$ is the sequence of substitutions occurring on ρ. Let T_1 and T_2 be two non-deterministic SSTs over two finite sets of variables $\mathcal{X}_1$ and $\mathcal{X}_2$ respectively, both with output variable O. Let $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) be the finite set of substitutions occurring in T_1 (resp. T_2). Let $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ and $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$. Let $\ell, k \in \mathbb{N}$.

We let $\mathbb{D}^{in}_{k,\ell,\mathcal{S}} = \{u \otimes \lambda \otimes \mu \mid u \in \Sigma^* \wedge \lambda \otimes \mu \in \mathbb{D}_{k,\ell,\mathcal{S}}\}$. The automaton recognizing $\mathbb{D}_{k,\ell,\mathcal{S}}$ from the proof of Theorem 5 can be easily extended into an automaton which recognizes $\mathbb{D}^{in}_{k,\ell,\mathcal{S}}$. Now, observe that T_1 is (k, ℓ)-included in T_2 iff $L(T_1) \subseteq \mathbb{D}^{in}_{k,\ell,\mathcal{S}}(L(T_2))$, where $\mathbb{D}^{in}_{k,\ell,\mathcal{S}}(L(T_2))$ denotes the set $\{u \otimes \lambda \mid \exists u \otimes \mu \in L(T_2) \colon u \otimes \lambda \otimes \mu \in \mathbb{D}^{in}_{k,\ell,\mathcal{S}}\}$. ◀

We turn to the variable minimization problem which is open for (deterministic) SSTs and undecidable for non-deterministic SSTs. We prove decidability for variable minimization up to fixed delay in both cases.

▶ **Theorem 9.** *Given integers k, ℓ, m and a non-deterministic SST T, it is decidable whether $T \equiv_{k,\ell} T'$ for some (non-deterministic resp. deterministic) SST T' that uses at most m variables.*

Proof sketch. First, we sketch the result for non-deterministic SST. Since we are looking for some non-deterministic SST T' with m variables such that $T \equiv_{k,\ell} T'$, we need to consider SSTs that produce at most $r := 2k + p$ letters (where p is the maximal number of letters produced by T in one step) per computation step in order to not violate the delay bound. The reasoning is that the difference between the output of the computations can be at most k letters, then in the next computation step, the computation that was k letters ahead may produce p letters, the other computation must recover the difference by producing at least p letters and at most $2k + p$ letters to keep the difference at most k. Thus, let $\mathcal{S} = \mathcal{S}_T \cup \mathcal{S}_{r,m}$ and $\mathcal{X} = \mathcal{X}_T \cup \mathcal{X}_m$, where $\mathcal{S}_T$ (resp. $\mathcal{X}_T$) are the substitutions (resp. variables) occurring in T, $\mathcal{X}_m = \{X_1, \cdots, X_m\}$, and $\mathcal{S}_{r,m}$ are substitutions over $\mathcal{X}_m$ producing at most r letters.

As as in the proof sketch of Theorem 8, $L(T)$ is the set of words of the form $u \otimes \tau(\rho)$, where $u \in \Sigma^*$, ρ is an accepting run of T over u, and $\tau(\rho)$ is the sequence of substitutions occurring on ρ. Furthermore, as in the proof sketch above, we let $\mathbb{D}^{in}_{k,\ell,\mathcal{S}} = \{u \otimes \lambda \otimes \mu \mid u \in \Sigma^* \wedge \lambda \otimes \mu \in \mathbb{D}_{k,\ell,\mathcal{S}}\}$. Now, observe that there exists some non-deterministic SST T' with m variables such that $T \equiv_{k,\ell} T'$ iff $L(T) \subseteq \mathbb{D}^{in}_{k,\ell,\mathcal{S}}(L)$ where L is the set of all words $u \otimes \mu$ such that $u \in \Sigma^*$ and $\mu \in \mathcal{S}^*_{r,m}$. As in the proof sketch above, $\mathbb{D}^{in}_{k,\ell,\mathcal{S}}(L)$ denotes the set $\{u \otimes \lambda \mid \exists u \otimes \mu \in L \colon u \otimes \lambda \otimes \mu \in \mathbb{D}^{in}_{k,\ell,\mathcal{S}}\}$.

If the goal is to have a deterministic SST, we need to adapt the above procedure. Let M be the projection of $\mathbb{D}^{in}_{k,\ell,\mathcal{S}}$ onto its first and third component. Hence, M contains all $u \otimes \mu$ such that there is some $u \otimes \lambda \in L(T)$ with $\mathsf{delay}_\ell(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$. We reduce the problem to a safety game played on a DFA for M. In alternation, one player provides an input letter, the other chooses a matching transition in the DFA. If the input player has provided a sequence $u \in \mathrm{dom}(R(T))$ then the play must be in an accepting state of the DFA, otherwise the output player has lost. We show that the output player has a winning strategy iff there exists an equivalent deterministic SST with at most m variables. ◀

Comparison with delay for finite transducers. We compare our notion of delay for streaming string transducers with the previously existing delay notion for finite transducers [24]. A *rational* SST is a non-deterministic SST with only one variable O, and updates all of the form $O := Ou$. In other words, a rational SST T is simply a finite transducer. Applying our delay notion to rational SSTs yields that if T_1 is (k, ℓ)-included in T_2 for two rational SSTs T_1, T_2, then T_1 is k-included in T_2 for the definition according to [24] and vice versa. The k-inclusion problem for finite transducers is PSPACE-complete for fixed k [24]. Hence, the complexity obtained in Theorem 8 matches this bound. on conceptual differences between those notions.

Consequences of completeness. We now turn to some consequences of our completeness result (Theorem 3). Inclusion for (deterministic) SSTs is known to be decidable [2]. It is undecidable for non-deterministic SSTs, but (k, ℓ)-inclusion is decidable (Theorem 8). Although Theorem 3 provides a new decision procedure for the inclusion problem for SSTs its value lies in showing that our notion of delay is a sensible approach to gain a better understanding of streaming string transducers. The following two corollaries are easy consequences of Theorem 3.

▶ **Corollary 10.** *Given an SST T and an integer m, there exist integers k, ℓ such that there exists an SST T' with m variables such that $T \equiv T'$ iff there exists an SST T'' with m variables such that $T \equiv_{k,\ell} T''$.*

Note that the above corollary does not imply that k and ℓ are *computable*. This would entail a solution for the variable minimization problem for SSTs which is open (and decidable for concatenation-free SSTs [6]). The next result is about rational functions, that is, functions recognizable by finite transducers.

▶ **Corollary 11.** *Given an SST T, there exist integers k, ℓ such that there exists a rational SST T' such that $T \equiv T'$ iff there exists a rational SST T'' such that $T \equiv_{k,\ell} T''$.*

It was shown that it decidable whether a deterministic two-way transducer (which is effectively equivalent to an SST) recognizes a rational function [23]. The decision procedure is effective, i.e., a finite transducer is constructed if possible. A procedure with improved complexity was given that yields a finite transducer of doubly exponential size in [7]. While computability is not implied by Corollary 11, note that one could compute k, ℓ satisfying the statement of Corollary 11 using Theorem 3 from an equivalent rational SST (if it exists) that has been obtained using the decision procedure from [7].

Other applications. We mention other potential applications of our delay notion that ought to be investigated. For instance, a *decomposition theorem* for SST relations is still only conjectured: Can every finite-valued SST relation be decomposed into a finite union of SST functions? In other settings where the corresponding statement holds (finite transducers [33], single-variable SST [25]), the main ingredients of the proof is the regularity and completeness of the appropriate notion of delay. So, having a good delay notion seems necessary to obtain such a result, but solving the decomposition theorem for SSTs does *not* seem to be a low-hanging fruit of our present study.

The notion of delay might also help to solve the *variable minimization* problem: Can we determine the minimal number of variables needed to define a given SST function? Corollary 10 makes some progress towards a positive answer, yet it remains to prove that the integers k and ℓ of the statement are computable, which is likely a complex problem. Another interesting research direction is to study how our notion of delay fares beyond SST. For *copyful* SST (where the copyless restriction of the substitutions is dropped), our notion of delay can be defined in the same manner, but its properties are unclear: our proofs of regularity and completeness both crucially rely on the copyless assumption.

References

1 Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.1.

2 Rajeev Alur and Pavol Cerný. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL 2011*, pages 599–610. ACM, 2011.

3 Rajeev Alur, Loris D'Antoni, and Mukund Raghothaman. Drex: A declarative language for efficiently evaluating regular string transformations. In Sriram K. Rajamani and David Walker, editors, *POPL 2015*, pages 125–137. ACM, 2015. doi:10.1145/2676726.2676981.

4 Rajeev Alur and Jyotirmoy V. Deshmukh. Nondeterministic streaming string transducers. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011. doi:10.1007/978-3-642-22012-8_1.

5 Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14–18, 2014*, pages 9:1–9:10. ACM, 2014. doi:10.1145/2603088.2603151.

6 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Minimizing resources of sweeping and streaming string transducers. In *ICALP 2016*, volume 55 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

7 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Untwisting two-way transducers in elementary time. In *LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

8 Nicolas Baudru and Pierre-Alain Reynier. From two-way transducers to regular function expressions. *Int. J. Found. Comput. Sci.*, 31(6):843–873, 2020. doi:10.1142/S0129054120410087.

9 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theor. Comput. Sci.*, 292(1):45–63, 2003. doi:10.1016/S0304-3975(01)00214-6.

10 Mikolaj Bojanczyk. Transducers with origin information. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2014. doi:10.1007/978-3-662-43951-7_3.

11 Mikolaj Bojanczyk. Polyregular functions. *CoRR*, abs/1810.08760, 2018. arXiv:1810.08760.

12 Sougata Bose. *On decision problems on word transducers with origin semantics. (Sur les problèmes de décision concernant les transducteurs de mots avec la sémantique d'origine)*. PhD thesis, University of Bordeaux, France, 2021. URL: https://tel.archives-ouvertes.fr/tel-03216029.

13 Sougata Bose, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. Origin-equivalence of two-way word transducers is in PSPACE. *CoRR*, abs/1807.08053, 2018. arXiv:1807.08053.

14 Christian Choffrut. Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977. doi:10.1016/0304-3975(77)90049-4.

15 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003. doi:10.1016/S0304-3975(01)00219-5.

16 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.

17 Luc Dartois, Emmanuel Filiot, and Nathan Lhote. Logics for word transductions with synthesis. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 295–304. ACM, 2018.

18 Luc Dartois, Paul Gastin, R. Govind, and Shankara Narayanan Krishna. Efficient construction of reversible transducers from regular transducer expressions. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2–5, 2022*, pages 50:1–50:13. ACM, 2022.

19 Luc Dartois, Paul Gastin, and Shankara Narayanan Krishna. Sd-regular transducer expressions for aperiodic transformations. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470738.

20 Laure Daviaud, Ismaël Jecker, Pierre-Alain Reynier, and Didier Villevalois. Degree of sequentiality of weighted automata. In Javier Esparza and Andrzej S. Murawski, editors, *FOSSACS 2017*, volume 10203 of *Lecture Notes in Computer Science*, pages 215–230, 2017. doi:10.1007/978-3-662-54458-7_13.

21 Calvin C. Elgot and Jorge E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, 1965. doi:10.1147/rd.91.0047.

22 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001. doi:10.1145/371316.371512.

23 Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. From two-way to one-way finite state transducers. In *LICS 2013*, pages 468–477. IEEE Computer Society, 2013.

24 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016*, volume 55 of *LIPIcs*, pages 125:1–125:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

25 Paul Gallot, Anca Muscholl, Gabriele Puppis, and Sylvain Salvati. On the decomposition of finite-valued streaming string transducers. In Heribert Vollmer and Brigitte Vallée, editors, *STACS 2017*, volume 66 of *LIPIcs*, pages 34:1–34:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.34.

26 Eitan M. Gurari. The equivalence problem for deterministic two-way sequential transducers is decidable. *SIAM J. Comput.*, 11(3):448–452, 1982. doi:10.1137/0211035.

27 Ismaël Jecker and Emmanuel Filiot. Multi-sequential word relations. In Igor Potapov, editor, *DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 288–299. Springer, 2015. doi:10.1007/978-3-319-21500-6_23.

28 Christophe Reutenauer and Marcel Paul Schützenberger. Minimization of rational word functions. *SIAM J. Comput.*, 20(4):669–685, 1991. doi:10.1137/0220042.

29 Jacques Sakarovitch and Rodrigo de Souza. On the decidability of bounded valuedness for transducers. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *MFCS 2008*, volume 5162 of *Lecture Notes in Computer Science*, pages 588–600. Springer, 2008. doi:10.1007/978-3-540-85238-4_48.

30 Jacques Sakarovitch and Rodrigo de Souza. On the decomposition of k-valued rational relations. In Susanne Albers and Pascal Weil, editors, *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, volume 1 of *LIPIcs*, pages 621–632. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2008. doi:10.4230/LIPIcs.STACS.2008.1324.

31 Jacques Sakarovitch and Rodrigo de Souza. Lexicographic decomposition of k-valued transducers. *Theory Comput. Syst.*, 47(3):758–785, 2010. doi:10.1007/s00224-009-9206-6.

32 Marcel Paul Schützenberger. A remark on finite transducers. *Inf. Control.*, 4(2-3):185–196, 1961. doi:10.1016/S0019-9958(61)80006-5.

33 Andreas Weber. Decomposing A k-valued transducer into k unambiguous ones. *RAIRO Theor. Informatics Appl.*, 30(5):379–413, 1996. doi:10.1051/ita/1996300503791.