TheoretiCS

Finite-valued Streaming String Transducers

Received Jun 10, 2024
Accepted Nov 10, 2024
Published Jan 08, 2025

Key words and phrases
streaming string transducers, finite valuedness

Emmanuel Filiot [a]
Ismaël Jecker [b]
Christof Löding [c]
Anca Muscholl [d]
Gabriele Puppis [e]
Sarah Winter [f]

a Université libre de Bruxelles, Belgium
b Université de Besançon, France
c RWTH Aachen University, Germany
d LaBRI, Université Bordeaux, France
e University of Udine, Italy
f IRIF, Université Paris Cité, France

ABSTRACT. A transducer is finite-valued if there exists a bound k such that any given input maps to at most k outputs. For classical one-way transducers, it is known since the 1980s that finite valuedness entails decidability of the equivalence problem. This result is in contrast with undecidability in the general case, making finite-valued transducers very appealing. For one-way transducers it is also known that finite valuedness itself is decidable and that any k-valued finite transducer can be decomposed into a union of k single-valued finite transducers.

In this article, we extend the above results to copyless streaming string transducers (SSTs), addressing open questions raised by Alur and Deshmukh in 2011. SSTs strictly extend the expressiveness of one-way transducers via additional variables that store partial outputs. We prove that any k-valued SST can be effectively decomposed into a union of k (single-valued) deterministic SSTs. As a corollary, we establish the equivalence between SSTs and two-way transducers in the finite-valued case, even though these models are generally incomparable. Another corollary provides an elementary upper bound for deciding equivalence of finite-valued SSTs. The equivalence problem was already known to be decidable, but the proof complexity relied on Ehrenfeucht's conjecture. Lastly, our main contribution shows that finite valuedness of SSTs is decidable, with the complexity being PSPACE in general, and PTIME when the number of variables is fixed.

Part of this work was initiated during the Dagstuhl seminar 23202 "Regular Transformations" [5]. A preliminary version of this article appeared at LICS 2024 [24].

Cite as Emmanuel Filiot, Ismaël Jecker, Christof Löding, Anca Muscholl, Gabriele Puppis, Sarah Winter. Finite-valued Streaming String Transducers. TheoretiCS, Volume 4 (2025), Article 1, 1-36.

https://theoretics.episciences.org
DOI 10.46298/theoretics.25.1

1. Introduction

Finite-state word transducers are simple devices that allow effective reasoning about data transformations. In their most basic form, they transform words using finite control. For example, the oldest transducer model, known as *generalized sequential machine*, extends deterministic finite state automata by associating each input with a corresponding output that is generated by appending finite words specified along the transitions. This rather simple model of transducer is capable of representing basic partial functions between words, e.g. the left rotating function $a_1\, a_2 \dots a_n \mapsto a_2 \dots a_n\, a_1$. Like automata, transducers can also be enhanced with non-determinism, as well as the ability of scanning the input several times (*two-wayness*). For example, the non-deterministic counterpart of generalized sequential machines, called here *one-way transducers*, can be used to represent the right rotating function $a_1 \dots a_{n-1}\, a_n \mapsto a_n\, a_1 \dots a_{n-1}$, but also word relations that are not partial functions, like for instance the relation that associates an input $a_1 \dots a_n$ with any output from $\{a_1\}^* \dots \{a_n\}^*$. Similarly, deterministic two-way transducers can compute the mirror function $a_1 \dots a_n \mapsto a_n \dots a_1$, the squaring function $w \mapsto ww$, etc.

Inspired by a logic-based approach applicable to arbitrary relational structures [18], *MSO-definable word transductions* were considered by Engelfriet and Hoogeboom [23] and shown to be equivalent to deterministic two-way transducers. Ten years later Alur and Cerný [6] proposed *streaming string transducers* (*SSTs* for short), a one-way model that uses write-only variables as additional storage. In SSTs, variables store strings and can be updated by appending or prepending strings, or concatenated together, but not duplicated (they are copyless). Alur and Cerný also showed that, in the functional case, that is, when restricting to transducers that represent partial functions, SSTs are equivalent to the model studied in [23], and thus in particular to two-way transducers. These equivalences between transducer models motivate nowadays the use of the term “regular” word function, in the spirit of classical results on regular word languages from automata theory and logics due to Büchi, Elgot, Trakhtenbrot, Rabin, and others.

While transducers inherit features like non-determinism and two-wayness from automata, these characteristics have an impact on their expressive power compared to automata. In the case of automata it is known that adding non-determinism and two-wayness does not affect the expressive power, as it only makes them more succinct in terms of number of states. It does not affect decidability of fundamental problems either, though succinctness makes some problems computationally harder. In contrast, for transducers, non-determinism and/or two-wayness significantly change the expressive power. For instance, non-deterministic one-way transducers may capture relations that are not partial functions, and thus not computable by generalized sequential machines[1]. This difference is also apparent at the level of decidability results. For example, the equivalence problem is in NLogSpace for generalized sequential machines, and undecidable for one-way transducers [27, 32, 41]. We should also mention that in the functional case it is possible to convert one transducer model to another (e.g. convert an SST to an equivalent two-way transducer). In the non-functional case, the picture is less satisfactory. In particular, non-deterministic SSTs and non-deterministic two-way transducers turn out to be incomparable: for example, the relation $\{(u\,v,\ v\,u) \,:\, u,v \in \Sigma^*\}$ can be represented in the former model but not in the latter, while the relation $\{(w,\ w^n) \,:\, w \in \Sigma^*, n \in \mathbb{N}\}$ can be represented in the latter model but not in the former. However, SSTs can still be converted to equivalent *non-deterministic MSO transductions* [9], which extend the original MSO transductions by existentially quantified monadic parameters.

There is however a class of relations that is close to (regular) word functions in terms of good behavior: the class of *finite-valued* relations. These are relations that associate a uniformly bounded number of outputs with each input. The concept of bounding the number of outputs associated with each input in a transducer is closely related to the notion of *finite ambiguity*, which refers to bounding the number of accepting runs. Ambiguity has been intensively studied in the context of formal languages, where it is shown, for instance, that equivalence of unambiguous automata is decidable in PTime [49]. In the context of relations, k-valuedness and k-ambiguity were initially considered in the setting of one-way transducers. For example, [29] showed that, for fixed k, one can decide in PTime whether a given one-way transducer is k-valued. Similarly, k-valuedness for fixed k can be decided in PSpace for two-way transducers and SSTs [9].

It is also clear that every k-ambiguous one-way transducer is k-valued. Conversely, it was shown that every k-valued one-way transducer can be converted to an equivalent, k-ambiguous one [51, 50, 45]. This result, even if it deals with a rather simple model of transducer, already uses advanced normalization techniques from automata theory and involves an exponential blow up in the number of states, as shown in the example below.

EXAMPLE 1.1. Fix $k \in \mathbb{N}$ and consider the relation

$$R_k \;=\; \big\{(w_1 \dots w_n,\ w_i) \,:\, n \in \mathbb{N},\ 1 \le i \le n,\ w_1, \dots, w_n \in \{0,1\}^k\big\}.$$

Examples of pairs in this relation, for $k = 2$, are $(00\,10\,11,\ 00)$, $(00\,10\,11,\ 10)$, and $(00\,10\,11,\ 11)$. For arbitrary k, the relation R_k associates at most 2^k outputs with each input (we say it is 2^k*-valued*). This relation can be realized by one-way transducers that exploit non-determinism to guess which block from the input becomes the output. For instance, a possible transducer T_k that realizes R_k repeatedly consumes blocks of k bits from the input, without outputting

1 Even if a non-deterministic one-way transducer computes a partial function, there may be no equivalent deterministic one-way transducer. However, this question can be decided in PTime [12, 3].

anything, until it non-deterministically decides to copy the next block, and after that it continues consuming the remaining blocks without output. Note that this transducer T_k has $O(k)$ states, it is finite-valued, but not finite-ambiguous, since the number of accepting runs per input depends on the number n of blocks in the input and it is thus unbounded. A finite-ambiguous transducer realizing the same relation R_k can be obtained at the cost of an exponential blow-up in the number of states, for instance by initially guessing and outputting a k-bit word w (this requires at least 2^k states), and then verifying that w occurs as a block of the input. ◆

Since k-ambiguous automata can be easily decomposed into a union of k unambiguous automata, the possibility of converting a k-valued one-way transducer to a k-ambiguous one entails a decomposition result of the following form: every k-valued one-way transducer is equivalent to a finite union of functional one-way transducers. One advantage of this type of decomposition is that it allows to generalize the decidability of the equivalence problem from functional to k-valued one-way transducers, which brings us back to the original motivation for considering classes of finite-valued relations. Decidability of the equivalence problem for k-valued one-way transducers was independently established in [33]. The latter work also states that the same techniques can be adapted to show decidability of equivalence for k-valued two-way transducers as well. Inspired by [33], the equivalence problem was later shown to be decidable also for k-valued SSTs [40]. However, the decidability results from [33] and [40] rely on the Ehrenfeucht conjecture [2, 28] and therefore provide no elementary upper bounds on the complexity.

Decomposing finite-valued SSTs and deciding finite valuedness for SSTs were listed as open problems in [9], more than 10 years ago, and represent our main contributions. Compared to one-way transducers, new challenges arise with SSTs, due to the extra power they enjoy to produce outputs. For example, consider the relation consisting of all pairs of the form $(w, 0^{n_0}1^{n_1})$ or $(w, 1^{n_1}0^{n_0})$, where $w \in \{0,1\}^*$, and n_b ($b = 0, 1$) is the number of occurrences of b in w. This relation is 2-valued, and is not realizable by any one-way transducer. On the other hand, the relation is realized by an SST T with a single state and two variables, denoted X_0, X_1 and both initially empty: whenever T reads $b \in \{0,1\}$, it non-deterministically applies the update $X_b := b\,X_b$ or $X_b := X_b\,b$ (while leaving X_{1-b} unchanged); at the end of the input, T outputs either $X_0\,X_1$ or $X_1\,X_0$. The ability of SSTs to generate outputs in a non-linear way makes their study challenging and intriguing. To illustrate this, consider a slight modification of T where X_0 is initialized with 1, instead of the empty word: the new SST is not finite-valued anymore, because upon reading 0^n it could output any word of the form $0^i\,1\,0^j$, with $i, j \in \mathbb{N}$ such that $i + j = n$.

Another open problem was to compare the expressive power of SSTs and two-way transducers in the finite-valued case. It is not hard to see that the standard translation from deterministic two-way transducers to deterministic SSTs also applies to the finite-valued case (cf. second part of the proof of Theorem 1.3). The converse translation, however, is far more complicated and relies on the decomposition theorem for SSTs which we establish in this article.

Contributions. The results presented in this article draw a rather complete picture about finite-valued SSTs, answering several open problems from [9]. First, we show that k-valued SSTs enjoy the same decomposition property as one-way transducers:

THEOREM 1.2. *For all $k \in \mathbb{N}$, every k-valued SST can be effectively decomposed into a union of k single-valued (or even deterministic) SSTs. The complexity of the construction is elementary.*

A first consequence of the above theorem is the equivalence of SSTs and two-way transducers in the finite-valued setting:

THEOREM 1.3. *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a finite-valued relation. If R can be realized by an SST, then an equivalent two-way transducer can be effectively constructed, and vice-versa.*

PROOF. If R is realized by an SST T, then we can apply Theorem 1.2 to obtain k unambiguous SSTs $T_1, \ldots, T_k$ whose union is equivalent to T. From [6] we know that in the functional case, SSTs and two-way transducers are equivalent. Thus, every T_i can be transformed effectively into an equivalent, even deterministic, two-way transducer. From this we obtain an equivalent k-ambiguous two-way transducer.

For the converse we start with a k-valued two-way transducer T and first observe that we can normalise T in such a way that the crossing sequences[2] of accepting runs of T are bounded by a constant linear in the size of T. Once we work with runs with bounded crossing sequences we can construct an equivalent SST in the same way as we do for deterministic two-way transducers. The idea is that during the run of the SST the variables record the outputs generated by the pieces of runs situated to the left of the current input position (see e.g. [38, 20] for self-contained proofs). ■

A second consequence of Theorem 1.2 is an elementary upper bound for the equivalence problem of finite-valued SSTs [40]:

THEOREM 1.4. *The equivalence problem for k-valued SSTs can be solved with elementary complexity.*

PROOF. Given two k-valued SSTs T, T', we first decompose them into unions of k deterministic SSTs $T_1, \ldots, T_k$ and $T'_1, \ldots, T'_k$, respectively. Finally, [9, Theorem 4.4] shows how to check the equivalence of $\bigcup_{i=1}^k T_i$ and $\bigcup_{i=1}^k T'_i$ in PSPACE. ■

Our last, and main, contribution establishes the decidability of finite valuedness for SSTs:

THEOREM 1.5. *Given any SST T, we can decide in* PSPACE *if T is finite-valued (and if the number of variables is fixed then the complexity is* PTIME*). Moreover, this problem is at least as hard as the equivalence problem for deterministic SSTs.*

---

2 A crossing sequence is a standard notion for finite-state two-way machines [48], and it is defined as the sequence of states in which a given input position is visited by an accepting run of the machine.

This last result is the most technical one, and requires to reason on particular substructures (W-patterns) of SSTs. Such substructures have been already used for one-way transducers, but for SSTs genuine challenges arise. The starting point of our proof is a recent result allowing to determine if two runs of an SST are far apart [25]. The proof then relies on identifying suitable patterns and extending techniques from word combinatorics to more involved word inequalities.

Based on the equivalence between SSTs and two-way transducers in the finite-valued setting (Theorem 1.3), and the decidability of finite valuedness for SST (Theorem 1.5), we exhibit an alternative proof for the following (known) result:

COROLLARY 1.6 ([53]). *Finite valuedness of two-way transducers is decidable in* PSPACE.

Observe also that without the results in this paper, the result of [53] could not help to show Theorem 1.5, because only the conversion from finite-valued two-way transducers to finite-valued SSTs was known (under the assumption that any input positions is visited a bounded number of times), but not the other way around. Also note that Theorems 1.2 and 1.3 together imply a decomposition result for finite-valued two-way transducers.

Similar results can be derived for non-deterministic MSO transductions. More precisely, since SSTs and non-deterministic MSO transductions are equivalent [9], Theorem 1.5 entails decidability of finite valuedness for non-deterministic MSO transductions as well. Moreover, since in the single-valued case, deterministic SSTs and MSO transductions are equivalent [6], Theorem 1.2 implies a decomposition result for MSO transductions: any k-valued non-deterministic MSO transduction can be decomposed as a union of k (deterministic) MSO transductions. Finally, from Theorem 1.3, we also obtain that, under the assumption of finite valuedness, non-deterministic MSO transductions, two-way transducers, and SSTs are equally expressive.

2. Preliminaries

For convenience, technical terms and notations in the electronic version of this manuscript are hyper-linked to their definitions (cf. `https://ctan.org/pkg/knowledge`).

Hereafter, $\mathbb{N}$ (resp. $\mathbb{N}_+$) denotes the set of non-negative (resp. strictly positive) integers, and Σ denotes a generic alphabet.

Words and relations. We denote by ε the empty word, by $|u|$ the length of a word $u \in \Sigma^*$, and by $u[i]$ its i-th letter, for $1 \leq i \leq |u|$. We introduce a convolution operation on words, which is particularly useful to identify robust and well-behaved classes of relations, as it is done for instance in the theory of automatic structures [13]. For simplicity, we only consider convolutions of words of the same length. Given $u, v \in \Sigma^*$, with $|u| = |v|$, the *convolution* $u \otimes v$ is a word over $(\Sigma^2)^*$ of length $|u| = |v|$ such that $(u \otimes v)[i] = (u[i], v[i])$ for all $1 \leq i \leq |u|$. For example,

$(aba) \otimes (bcc) = (a,b)(b,c)(a,c)$. As $\otimes$ is associative, we may write $u \otimes v \otimes w$ for any words u, v, w.

A relation $R \subseteq (\Sigma^*)^k$ is *length-preserving* if $|u_1| = \cdots = |u_k|$ for all $(u_1, \ldots, u_k) \in R$. A length-preserving relation is *automatic* if the language $\{u_1 \otimes \ldots \otimes u_k \mid (u_1, \ldots, u_k) \in R\}$ is recognized by a finite state automaton. A binary relation $R \subseteq \Sigma^* \times \Sigma^*$ (not necessarily length-preserving) is *k-valued*, for $k \in \mathbb{N}$, if for all $u \in \Sigma^*$, there are at most k words v such that $(u, v) \in R$. It is *finite-valued* if it is k-valued for some k.

Variable updates. Fix a finite set of variables $\mathcal{X} = \{X_1, \ldots, X_m\}$, disjoint from the alphabet Σ. A (copyless) *update* is any mapping $\alpha : \mathcal{X} \to (\Sigma \uplus \mathcal{X})^*$ such that each variable $X \in \mathcal{X}$ appears *at most once* in the word $\alpha(X_1) \ldots \alpha(X_m)$. Such an update can be morphically extended to words over $\Sigma \uplus \mathcal{X}$, by simply letting $\alpha(a) = a$ for all $a \in \Sigma$. Using this, we can compose any two updates α, β to form a new update $\alpha\,\beta : \mathcal{X} \to (\Sigma \uplus \mathcal{X})^*$, defined by $(\alpha\,\beta)(X) = \beta(\alpha(X))$ for all $X \in \mathcal{X}$. An update is called *initial* (resp. *final*) if all variables in $\mathcal{X}$ (resp. $\mathcal{X} \setminus \{X_1\}$) are mapped to the empty word. The designated variable X_1 is used to store the final output produced by an SST, as defined in the next paragraph.

Streaming string transducers. A (non-deterministic, copyless) *streaming string transducer* (*SST* for short) is a tuple $T = (\Sigma, \mathcal{X}, Q, Q_{\mathrm{init}}, Q_{\mathrm{final}}, O, \Delta)$, where Σ is an alphabet, $\mathcal{X}$ is a finite set of variables, Q is a finite set of states, $Q_{\mathrm{init}}, Q_{\mathrm{final}} \subseteq Q$ are the sets of initial and final states, O is a function from final states to final updates, and Δ is a finite transition relation consisting of tuples of the form (q, a, α, q'), where $q, q' \in Q$ are the source and target states, $a \in \Sigma$ is an input symbol, and α is an update. We often denote a transition $(q, a, \alpha, q') \in \Delta$ by the annotated arrow:

$$q \xrightarrow{a/\alpha} q'.$$

The *size* $|T|$ of an SST T is defined as the number of states plus the size of its transition relation.

A *run* of T is a sequence of transitions from Δ of the form

$$\rho = q_0 \xrightarrow{a_1/\alpha_1} q_1 \xrightarrow{a_2/\alpha_2} q_2 \ldots q_{n-1} \xrightarrow{a_n/\alpha_n} q_n.$$

The *input consumed by* ρ is the word $\mathrm{in}(\rho) = a_1 \ldots a_n$. The *update induced by* ρ is the composition $\beta = \alpha_1 \ldots \alpha_n$. We write $\rho : u/\beta$ to mean that ρ is a run with u as consumed input and β as induced update. A run ρ as above is *accepting* if the first state is initial and the last state is final, namely, if $q_0 \in Q_{\mathrm{init}}$ and $q_n \in Q_{\mathrm{final}}$. In this case, the induced update, extended to the left with the initial update denoted by ι and to the right with the final update $O(q_n)$, gives rise to an update $\iota\,\beta\,O(q_n)$ that maps X_1 to a word over Σ and all remaining variables to the empty word. In particular, the latter update determines the *output produced by* ρ, defined as the word $\mathrm{out}(\rho) = (\iota\,\beta\,O(q_n))(X_1)$.

The *relation realized by* an SST T is

$$\mathscr{R}(T) = \{(\mathrm{in}(\rho), \mathrm{out}(\rho)) \in \Sigma^* \times \Sigma^* \mid \rho \text{ accepting run of } T\}$$

An SST is *k-valued* (resp. *finite-valued*) if its realized relation is so. It is *deterministic* if it has a single initial state and the transition relation is a partial function (from pairs of states and input letters to pairs of updates and states). It is *unambiguous* if it admits at most one accepting run on each input. Similarly, it is called *k-ambiguous* if it admits at most k accepting runs on each input. Of course, every deterministic SST is unambiguous, and every unambiguous SST is single-valued (i.e. 1-valued). Two SSTs T_1, T_2 are *equivalent* if $\mathscr{R}(T_1) = \mathscr{R}(T_2)$. The equivalence problem for SSTs is undecidable in general, and it is so even for one-way transducers [27, 32]. However, decidability is recovered for finite-valued SSTs:

THEOREM 2.1 ([40]). *The equivalence problem for finite-valued SSTs is decidable.*

Note that checking equivalence is known to be in PSPACE for *deterministic* SSTs. This easily generalizes to unions of deterministic (hence single-valued) SSTs, because the equivalence checking algorithm is exponential only in the number of variables:

THEOREM 2.2 ([9]). *The following problem is in* PSPACE: given $n+m$ deterministic SSTs $T_1, \ldots, T_n, T'_1, \ldots, T'_m$, decide whether $\bigcup_{i=1}^{n} \mathscr{R}(T_i) = \bigcup_{j=1}^{m} \mathscr{R}(T'_j)$.

For any fixed k, the k-valuedness property is decidable in PSPACE:

THEOREM 2.3 ([9]). *For any fixed $k \in \mathbb{N}$, the following problem is in*[3] PSPACE: given an SST T, decide whether T is k-valued. It is in PTIME if one further restricts to SSTs with a fixed number of variables.

The decidability status of finite valuedness for SSTs, i.e., if k is unknown, was an open problem. Part of our contribution is to show that this problem is decidable, too.

2.1 Pumping and word combinatorics

When reasoning with automata, it is common practice to use pumping arguments. This section introduces pumping for SSTs, as well as combinatorial results for reasoning about (in)equalities between pumped outputs of SSTs.

In order to have adequate properties for pumped runs of SSTs, the notion of loop needs to be defined so as to take into account how the content of variables "flows" into other variables

3 In [9], no complexity result is provided, but the decidability procedure relies on a reduction to the emptiness of a 1-reversal $k(k+1)$-counter machine, based on the proof for equivalence of deterministic SST [7]. The counter machine is exponential in the number of variables only. The result follows since the emptiness problem for counter machines with fixed number of reversals and fixed number of counters is in NLOGSPACE [30].

when performing an update. We define the *skeleton* of an update $\alpha : \mathcal{X} \to (\Sigma \uplus \mathcal{X})^*$ as the update $\hat{\alpha} : \mathcal{X} \to \mathcal{X}^*$ obtained from α by removing all the letters of Σ from the right-hand side. Note that there are only finitely many skeletons, and their composition forms a finite monoid, called the *skeleton monoid* (this notion is very similar to the flow monoid from [40], but does not rely on any normalization).

A *loop* of a run ρ of an SST is any factor L of ρ that starts and ends in the same state and induces a *skeleton-idempotent* update, namely, an update α such that α and $\alpha\,\alpha$ have the same skeleton. For example, the update $\alpha : \ X_1 \mapsto a\,X_1\,b\,X_2\,c,\ X_2 \mapsto a$ is skeleton-idempotent and thus can be part of a loop. A loop in a run will be denoted by an interval $[i, j]$. In this case, it is convenient to assume that the indices i, j represent "positions" in-between the transitions, thus identifying occurrences of states; in this way, adjacent loops can be denoted by intervals of the form $[i_1, i_2]$, $[i_2, i_3]$, etc. In particular, if the run consists of n transitions, then the largest possible interval on it is $[0, n]$. For technical reasons, we do allow *empty* loops, that is, loops of the form $[i, j]$, with $i = j$ and with the induced update being the identity function on $\mathcal{X}$.

The run obtained from ρ by pumping n times a loop L is denoted $\mathrm{pump}_L^n(\rho)$. If we are given an m-tuple of *pairwise disjoint* loops $\bar{L} = (L_1, \ldots, L_m)$ and an m-tuple of (positive) numbers $\bar{n} = (n_1, \ldots, n_m)$, then we write $\mathrm{pump}_{\bar{L}}^{\bar{n}}(\rho)$ for the run obtained by pumping simultaneously n_i times L_i, for each $1 \leq i \leq m$.

The next lemma is a Ramsey-type argument that, based on the number of states of the SST, the size of the skeleton monoid, and a number n, derives a minimum length for a run to witness $n + 1$ points and loops between pairs of any of these points. The reader can refer to [35] to get good estimates of the values of E, H.

LEMMA 2.4. *Given an SST, one can compute two numbers E, H such that for every run ρ, every $n \in \mathbb{N}$, and every set $I \subseteq \{0, \ldots, |\rho|\}$ of cardinality $En^H + 1$, there is a subset $I' \subseteq I$ of cardinality $n + 1$ such that for all $i < j \in I'$ the interval $[i, j]$ is a loop of ρ. The values of E, H are elementary in the size of the SST.*

PROOF. The article [35] shows for any given monoid M a bound $R_M(n)$ with the property that any sequence from M^* of length larger than $R_M(n)$ contains n consecutive infixes such that for some idempotent e of M (i.e., satisfying $ee = e$) each such infix multiplies out to e. In our case, the monoid M is the product of the monoid $((Q \times Q) \cup \{0\}, \cdot)$ with $(p, q) \cdot (q, r) = (p, r)$ (resp. $(p, q) \cdot (q', r) = 0$ if $q \neq q'$) and the skeleton monoid of the SST. Thus, any infix of the run that multiplies out to an idempotent (after mapping each transition to the corresponding monoid element) corresponds to a loop of the SST. The upper bound $R_M(n)$ is exponential only in the size of M (cf. Theorem 1 in [35]). ■

Below, we describe the effect on the output of pumping loops in a run of an SST. We start with the following simple combinatorial result:

LEMMA 2.5. *Let α be a skeleton-idempotent update. For every variable X, there exist two words $u, v \in \Sigma^*$ such that, for all positive natural numbers $n \in \mathbb{N}_+$, $\alpha^n(X) = u^{n-1}\,\alpha(X)\,v^{n-1}$.*

PROOF. Let $\hat{\alpha}$ be the idempotent skeleton of α. We first prove the following claim:

CLAIM. *For all $X \in \mathcal{X}$, if $\hat{\alpha}(X) \neq \varepsilon$, then X occurs in $\alpha(X)$.*

PROOF OF THE CLAIM. The proof is by induction on the number of variables X such that $\hat{\alpha}(X) \neq \varepsilon$. The base case holds vacuously. As for the induction step, fix a variable X_0 such that $\hat{\alpha}(X_0) \neq \varepsilon$. We distinguish two cases:

- If X_0 occurs in $\alpha(X_0)$, then the claim clearly holds for X_0 and it remains to prove it for all other variables $Y \in \mathcal{X} \setminus \{X_0\}$. Since α is copyless, X_0 does not occur in $\alpha(Y)$, for all $Y \in \mathcal{X} \setminus \{X_0\}$. Therefore, the restricted update $\alpha|_{\mathcal{X}\setminus\{X_0\}}$ has an idempotent skeleton, and by the inductive hypothesis it satisfies the claim. From this we immediately derive that the claim also holds for the original update α.
- If X_0 does not occur in $\alpha(X_0)$, then $\alpha(X_0)$ still contains at least one occurrence of another variable, since $\hat{\alpha}(X_0) \neq \varepsilon$. So, suppose that $\alpha(X_0) = h\,Y\,t$ for some $h, t \in (\mathcal{X} \cup \Sigma)^*$ and $Y \in \mathcal{X}$. Since α is skeleton-idempotent, we have $(\alpha\alpha)(X_0) = \alpha(h)\,\alpha(Y)\,\alpha(t) = h'\,Y\,t'$ for some $h', t' \in (\mathcal{X} \cup \Sigma^*)$. Now, if Y occurs in $\alpha(Y)$, then we can apply the inductive hypothesis on the restriction $\alpha|_{\mathcal{X}\setminus\{Y\}}$, as in the previous case, reaching a contradiction for X_0. Otherwise, if Y does not occur in $\alpha(Y)$, we also reach a contradiction by arguing as follows. Since Y occurs in $\alpha(h)\,\alpha(Y)\,\alpha(t)$ but not in $\alpha(Y)$, then it occurs in $\alpha(h)\,\alpha(t)$. Since X_0 is the unique variable such that Y occurs in $\alpha(X_0)$ (because α is copyless), necessarily X_0 occurs in $h\,t$ and hence in $\alpha(X_0)$ as well, which contradicts the initial assumption. ■

We conclude the proof of the lemma. Let $X \in \mathcal{X}$. If $\alpha(X)$ does not contain any variable, then we immediately get the result, as $\alpha^n(X) = \alpha(X)$ for all $n \geq 1$. Otherwise, if $\alpha(X)$ contains some variable, then we know that $\hat{\alpha}(X) \neq \varepsilon$, so by the above claim, X occurs in $\alpha(X)$. Hence, $\alpha(X) = h\,X\,t$ for some $h, t \in (\mathcal{X} \cup \Sigma)^*$. We now prove that $\alpha(h)\,\alpha(t) \in \Sigma^*$. Indeed, let Y be any variable in $h\,t$ (if there is no such variable then clearly $\alpha(h)\,\alpha(t)$ contains no variable either). Since α is copyless, Y does not occur in $\alpha(Y)$, hence by the above claim (contrapositive), $\hat{\alpha}(Y) = \varepsilon$, hence $\alpha(Y) \in \Sigma^*$. We then derive $(\alpha\alpha)(X) = \alpha(h)\,\alpha(X)\,\alpha(t)$, where $\alpha(h), \alpha(t) \in \Sigma^*$, and so we can take $u = \alpha(h)$ and $v = \alpha(t)$. To conclude, we have $\alpha^2(X) = u\,\alpha(X)\,v$, so for $n > 2$, $\alpha^n(X) = \alpha^{n-2}(\alpha^2(X)) = \alpha^{n-2}(u\,\alpha(X)\,v) = u\,\alpha^{n-1}(X)\,v$, as claimed. ■

It follows that pumping loops in a run corresponds to introducing repeated copies of factors in the output. Similar results can be found in [40] for SSTs and in [42, 22] for two-way transducers:

COROLLARY 2.6. *Let ρ be an accepting run of an SST and let $\bar{L} = (L_1, \ldots, L_m)$ be a tuple of pairwise disjoint loops in ρ. Then, for some $r \leq 2m|\mathcal{X}|$ there exist words $w_0, \ldots, w_r, u_1, \ldots, u_r$ and* *indices* $1 \le i_1, \ldots, i_r \le m$*, not necessarily distinct, such that for every tuple* $\bar{n} = (n_1, \ldots, n_m) \in \mathbb{N}_+^m$ *of positive natural numbers,*

$$\mathrm{out}(\mathrm{pump}_L^{\bar{n}}(\rho)) \;=\; w_0\, u_1^{n_{i_1}-1}\, w_1\, \ldots\, u_r^{n_{i_r}-1}\, w_r.$$

PROOF. This follows immediately from Lemma 2.5. Note that the content of any variable X just after pumping a loop either appears as infix of the final output, or is erased by some later update. In both cases, each pumped loop L_i induces in the output $(n_i - 1)$-folded repetitions of $2k$ (possibly empty) factors, where k is the number of variables of the SST. Since the loops are pairwise disjoint, they contribute such factors without any interference. The final output $\mathrm{out}(\mathrm{pump}_L^{\bar{n}}(\rho))$ thus features repetitions of $r = 2km$ (possibly empty) factors. ■

The rest of the section analyses properties of words with repeated factors like the one in Corollary 2.6.

DEFINITION 2.7. A *word inequality* with repetitions parametrized in $\mathtt{X}$ is a pair $e = (w, w')$ of terms of the form

$$\begin{aligned} w &= s_0\, t_1^{\mathrm{x}_1}\, s_1\, \ldots\, t_m^{\mathrm{x}_m}\, s_m \\ w' &= s'_0\, t'^{\,\mathrm{y}_1}_1\, s'_1\, \ldots\, t'^{\,\mathrm{y}_n}_n\, s'_n \end{aligned}$$

where $s_i, t_i, s'_j, t'_j \in \Sigma^*$ and $\mathrm{x}_i, \mathrm{y}_j \in \mathtt{X}$ for all i, j. The set of *solutions* of $e = (w, w')$, denoted $\mathrm{Sols}(e)$, consists of the mappings $f : \mathtt{X} \to \mathbb{N}$ such that $f(w) \neq f(w')$, where $f(w)$ is the word obtained from w by substituting every formal parameter $\mathrm{x} \in \mathtt{X}$ by $f(\mathrm{x})$, and similarly for $f(w')$. A *system of word inequalities* is a non-empty finite set E of inequalities as above, and its set of solutions is given by $\mathrm{Sols}(E) = \bigcap_{e \in E} \mathrm{Sols}(e)$.

The next theorem states that if there exists a solution to a system of inequalities parameterized by a single variable x, then the set of solutions is co-finite.

THEOREM 2.8 ([43, Theorem 4.3]). *Given a word inequality e with repetitions parameterized by single variable* x*,* $\mathrm{Sols}(e)$ *is either empty or co-finite; more precisely, if the left (resp. right) hand-side of e contains m (resp. n) repeating patterns (as in Definition 2.7), then either* $\mathrm{Sols}(e) = \emptyset$ *or* $|\mathbb{N} \setminus \mathrm{Sols}(e)| \le m + n$*.*

Finally, we present two corollaries of the above theorem, that will be used later. The first corollary concerns satisfiability of a system of inequalities. Formally, we say that a word inequality e (resp. a system of inequalities E) is *satisfiable* if its set of solutions is non-empty.

COROLLARY 2.9. *Let E be a finite system of word inequalities. If every inequality* $e \in E$ *is satisfiable, then so is the system E.*

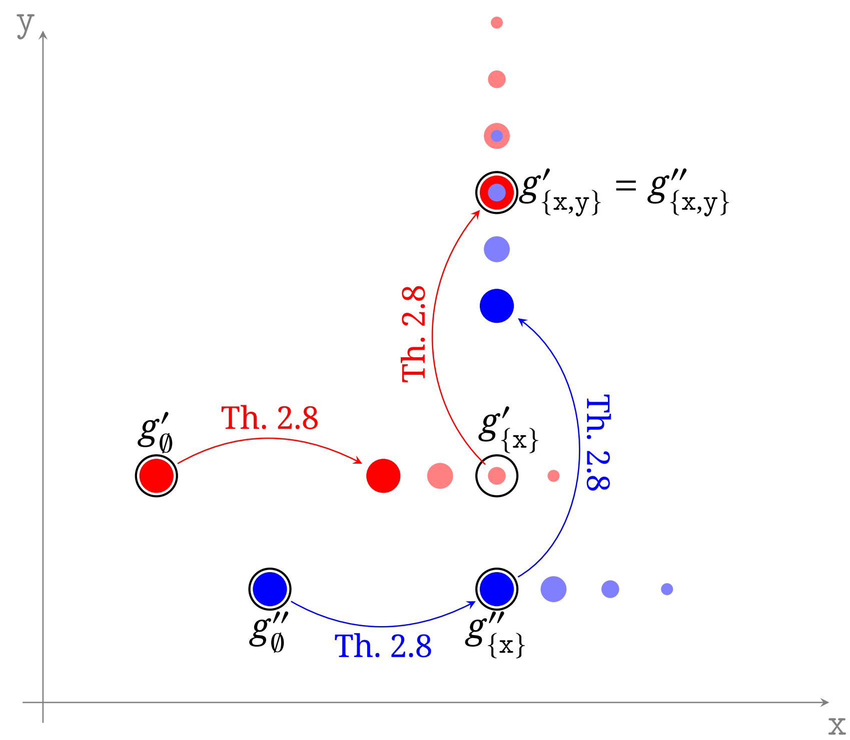


Figure 1. Illustration of an argument for the proof of Corollary 2.9

PROOF. All inequalities considered hereafter have parameters in $\mathtt{X} = \{\mathrm{x}_1, \ldots, \mathrm{x}_k\}$. We are going to compare functions from $\mathtt{X}$ to $\mathbb{N}$ based on suitable partial orders, each parametrized by a variable. Formally, given two functions $f, g : \mathtt{X} \to \mathbb{N}$ and a variable $\mathrm{x} \in \mathtt{X}$, we write $f \leq_{\mathrm{x}} g$ iff $f(\mathrm{x}) \leq g(\mathrm{x})$ and $f(\mathrm{y}) = g(\mathrm{y})$ for all $\mathrm{y} \in \mathtt{X} \setminus \{\mathrm{x}\}$. We prove the following two properties (the first property is equivalent to the claim of the lemma):

$$\bigwedge_{e \in E} \exists f \in \mathsf{Sols}(e) \quad \longrightarrow \quad \exists g \in \mathsf{Sols}(E) \tag{1}$$

$$\forall f \in \mathsf{Sols}(E) \;\; \forall \mathrm{x} \in \mathtt{X} \;\; \exists g \geq_{\mathrm{x}} f \;\; \forall h \geq_{\mathrm{x}} g \quad : \quad h \in \mathsf{Sols}(E) \tag{2}$$

The proof goes by double induction on the cardinality of E and the number k of parameters.

The base case is when E has cardinality 1. In this case Property (1) holds trivially. We see how Property (2) follows from Theorem 2.8. Let $E = \{e\}$, $f \in \mathsf{Sols}(e)$, and fix an arbitrary variable $\mathrm{x} \in \mathtt{X}$. We construct the inequality e' with x as single formal parameter, by instantiating in e every other parameter $\mathrm{y} \in \mathtt{X} \setminus \{\mathrm{x}\}$ with the value $f(\mathrm{y})$. By construction, f restricted to $\{\mathrm{x}\}$ is a solution of e', and thus, by Theorem 2.8, $\mathsf{Sols}(e')$ is co-finite. This means that there is a number $x_0 \in \mathbb{N}$ such that $x_0 \geq f(\mathrm{x})$ and, for all $x_1 \geq x_0$, the mapping $\mathrm{x} \mapsto x_1$ is a solution to e' as well. This property can be transferred to the original inequality e, as follows. We define $g = f[\mathrm{x}/x_1]$ as the function obtained from f by replacing the image of x with x_1. Note that $g \geq_{\mathrm{x}} f$ and, for all $h \geq_{\mathrm{x}} g$, $h \in \mathsf{Sols}(e)$. This proves Property (2).

As for the inductive step, suppose that E is a system with at least two inequalities, and divide E into two sub-systems, E' and E'', with cardinalities strictly smaller than that of E.

Let us first prove Property (1) for E. Suppose that every inequality in E is satisfiable. By the inductive hypothesis, E' and E'' are also satisfiable; in particular, there exist solutions g' and g'' of E' and E'', respectively. We proceed with a second induction to prove that, for larger and larger sets of variables $\mathtt{Y} \subseteq \mathtt{X}$, there are solutions of E' and E'' that agree on all the variables from $\mathtt{Y}$, namely:

$$\exists g'_{\mathtt{Y}} \in \mathrm{Sols}(E') \;\; \exists g''_{\mathtt{Y}} \in \mathrm{Sols}(E'') \;\; \forall \mathtt{y} \in \mathtt{Y} \;\; g'_{\mathtt{Y}}(\mathtt{y}) = g''_{\mathtt{Y}}(\mathtt{y}) \;. \tag{$\star$}$$

Of course, for $\mathtt{Y} = \mathtt{X}$, the above property will imply the existence of a solution of E. The reader can refer to Figure 1 as an illustration of the arguments that follow (axes correspond to variables, and red and blue dots represent solutions of the systems E' and E'', respectively).

For $\mathtt{Y} = \emptyset$, the claim ($\star$) is trivial, since we can simply let $g'_{\emptyset} = g'$ and $g''_{\emptyset} = g''$. For the inductive step, suppose that ($\star$) holds for $\mathtt{Y}$ and let us prove it also holds for $\mathtt{Y}' = \mathtt{Y} \uplus \{\mathtt{x}\}$. By inductive hypothesis, E' and E'' satisfy Property (2). In particular, by instantiating f with $g'_{\mathtt{Y}}$ (resp. $g''_{\mathtt{Y}}$) in Property (2), we obtain the existence of $g' \geq_{\mathtt{x}} g'_{\mathtt{Y}}$ such that, for all $h' \geq_{\mathtt{x}} g'$, $h' \in \mathrm{Sols}(E')$ (resp. $g'' \geq_{\mathtt{x}} g''_{\mathtt{Y}}$ such that, for all $h'' \geq_{\mathtt{x}} g''$, $h'' \in \mathrm{Sols}(E'')$). Note that the functions $g'_{\mathtt{Y}}, g''_{\mathtt{Y}}, g', g''$ all agree on the variables from $\mathtt{Y}$. Moreover, without loss of generality, we can assume that g' and g'' also agree on the variable $\mathtt{x}$: indeed, if this were not the case, we could simply replace the $\mathtt{x}$-images of g' and g'' with $\max\{g'(\mathtt{x}), g''(\mathtt{x})\}$, without affecting the previous properties. Property ($\star$) now follows from letting $g'_{\mathtt{Y}'} = g'$ and $g''_{\mathtt{Y}'} = g''$. This concludes the proof of the inductive step for Property (1).

Let us now prove the inductive step for Property (2). Let f be a solution of E and let $\mathtt{x} \in \mathtt{X}$. Since both E' and E'' satisfy Property (2) and since $f \in \mathrm{Sols}(E') \cap \mathrm{Sols}(E'')$, there are $g', g'' \geq_{\mathtt{x}} f$ such that, for all $h' \geq_{\mathtt{x}} g'$ and $h'' \geq_{\mathtt{x}} g''$, $h' \in \mathrm{Sols}(E')$ and $h'' \in \mathrm{Sols}(E'')$. Since $g', g'' \geq_{\mathtt{x}} f$, g' and g'' agree on all variables, except possibly $\mathtt{x}$. Without loss of generality, we can also assume that g' and g'' agree on $\mathtt{x}$: as before, if this were not the case, we could replace the $\mathtt{x}$-images of g' and g'' with $\max\{g'(\mathtt{x}), g''(\mathtt{x})\}$, while preserving the previous properties. Now that we have $g' = g''$, we can use this function to witness Property (2), since, for all $h \geq_{\mathtt{x}} g'$ $(= g'')$, we have $h \in \mathrm{Sols}(E') \cap \mathrm{Sols}(E'') = \mathrm{Sols}(E)$. ■

The second corollary is related to the existence of large sets of solutions for a satisfiable word inequality that avoid any correlation between variables. To formalize the statement, it is convenient to fix a total order on the variables of the inequality, say $\mathtt{x}_1, \ldots, \mathtt{x}_k$, and then identify every function $f : \mathtt{X} \to \mathbb{N}$ with the k-tuple of values $\bar{x} = (x_1, \ldots, x_k)$, where $x_i = f(\mathtt{x}_i)$ for all $i = 1, \ldots, k$. According to this correspondence, the corollary states the existence of sets of solutions that look like Cartesian products of finite intervals of values, each with arbitrarily large cardinality. The statement of the corollary is in fact slightly more complicated than this, as it discloses dependencies between the intervals. We also observe that the order in which we

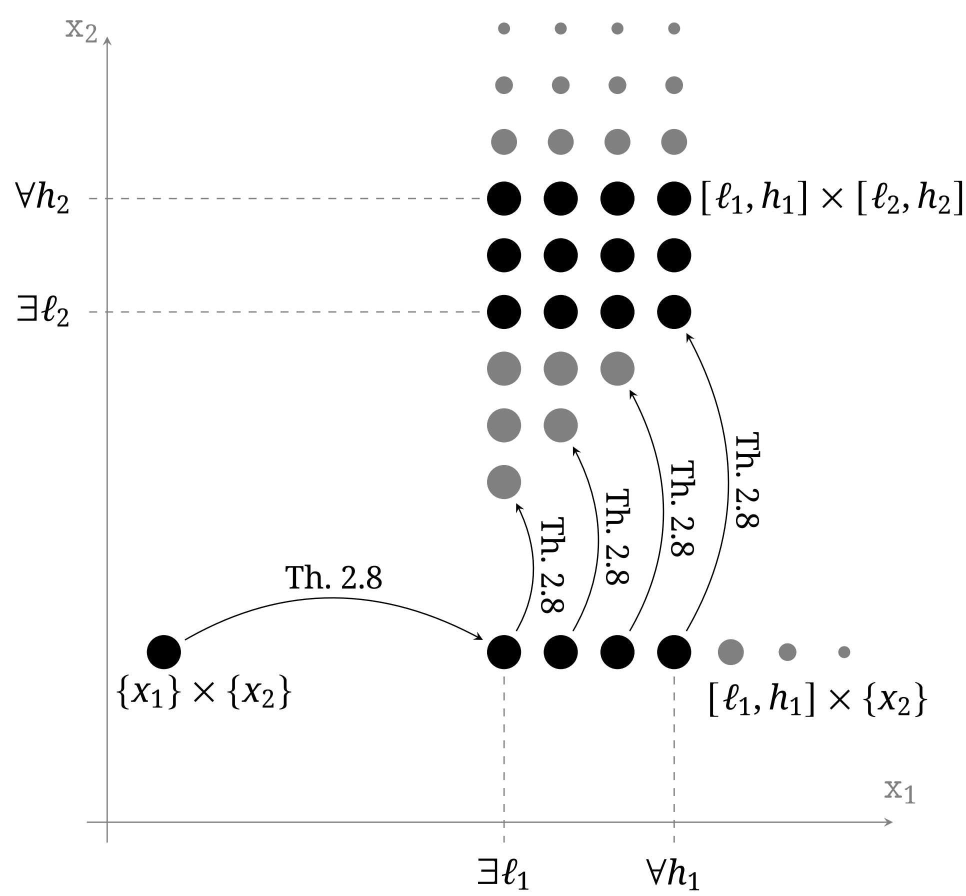


Figure 2. Illustration of an argument for the proof of Corollary 2.10

list the variables is arbitrary, but different orders will induce different dependencies between intervals.

COROLLARY 2.10. *Let e be a word inequality with repetitions parametrized in* $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$. *If e is satisfiable, then*

$$\exists \ell_1 \, \forall h_1 \; \ldots \; \exists \ell_k \, \forall h_k \quad \underbrace{[\ell_1, h_1]}_{\textit{values for } \mathtt{x}_1} \times \ldots \times \underbrace{[\ell_k, h_k]}_{\textit{values for } \mathtt{x}_k} \subseteq \mathrm{Sols}(e).$$

PROOF. Let e be a satisfiable word inequality parametrized in $\mathtt{X}$ and let $\bar{x} = (x_1, \ldots, x_k)$ be any solution of e. We will prove the following claim by induction on $i = 0, \ldots, k$:

$$\exists \ell_1 \, \forall h_1 \; \ldots \; \exists \, \ell_i \, \forall h_i \quad [\ell_1, h_1] \times \cdots \times [\ell_i, h_i] \times \{x_{i+1}\} \times \cdots \times \{x_k\} \subseteq \mathrm{Sols}(e) \tag{$\star$}$$

Note that for $i = k$ the above claim coincides with the statement of the corollary. The reader can also refer to Figure 2 as an illustration of the arguments that follow (axes correspond to variables $\mathtt{x}_1$ and $\mathtt{x}_2$, dots represent generic solutions of e, clusters of black dots represent solutions in the form of Cartesian products, like those that appear in ($\star$).

For the base case $i = 0$, the claim ($\star$) is vacuously true, as $\bar{x} = (x_1, \ldots, x_k)$ is a solution of e.

For the inductive step, we need to show that if $(\star)$ holds for $i < k$, then it also holds for $i+1$. It is in fact sufficient to prove that, for $i < k$,

$$[\ell_1, h_1] \times \cdots \times [\ell_i, h_i] \times \{x_{i+1}\} \times \cdots \times \{x_k\} \subseteq \mathrm{Sols}(e)$$

implies

$$\exists \ell_{i+1}\ \forall h_{i+1}$$
$$[\ell_1, h_1] \times \cdots \times [\ell_{i+1}, h_{i+1}] \times \{x_{i+2}\} \times \cdots \times \{x_k\} \subseteq \mathrm{Sols}(e).$$

For brevity, we let $S = [\ell_1, h_1] \times \cdots \times [\ell_i, h_i] \times \{x_{i+1}\} \times \cdots \times \{x_k\}$, and we assume that $S \subseteq \mathrm{Sols}(e)$. For every tuple $\bar{s} \in S$, we consider the word inequality $e_{\bar{s}}$ over a single variable x_{i+1} that is obtained from e by instantiating every other variable x_j $(j \neq i)$ with $\bar{s}[j]$. Since $S \subseteq \mathrm{Sols}(e)$, we know that $e_{\bar{y}}$ is satisfiable, and hence by Theorem 2.8, $e_{\bar{s}}$ has co-finitely many solutions. This means that there is $\ell_{\bar{s}}$ such that, for all $x' \geq \ell_{\bar{s}}$, x' is also a solution of $e_{\bar{s}}$. This property can be transferred to our original inequality e:

CLAIM. *For every $\bar{s} \in S$, there is $\ell_{\bar{s}}$ such that, for every $x' \geq \ell_{\bar{s}}$, the tuple $\bar{s}[i+1 \mapsto x']$, obtained from $\bar{s}$ by replacing the $(i+1)$-th value with x', is a solution of e.*

Now, the existentially quantified value ℓ_{i+1} can be set to the maximum of the $\ell_{\bar{s}}$'s, for all $\bar{s} \in S$ (for this definition to make sense, it is crucial that the set S is finite). In this way, thanks to the previous claim, the containment $[\ell_1, h_1] \times \cdots \times [\ell_{i+1}, h_{i+1}] \times \{x_{i+2}\} \times \cdots \times \{x_k\} \subseteq \mathrm{Sols}(e)$ holds for all choices of the universally quantified value h_{i+1}. This proves the inductive step for $(\star)$ from i to $i+1$. ∎

2.2 Delay between accepting runs

We briefly recall the definitions from [25], that introduces a measure of similarity (called delay) between accepting runs of an SST that have the same input and the same output.

We first give some intuition, followed by definitions and an example. Naturally, the difference between the amount of output symbols produced during a run should be an indicator of (dis)similarity. However, as SSTs do not necessarily build their output from left to right, one must also take into account the position where an output symbol is placed. For example, compare two runs ρ and ρ' on the same input that produce the same output *aaabbb*. After consuming a prefix of the input, ρ may have produced $aaa\_\_\_$ and ρ' may have produced $\_\_\_bbb$. The amount of produced output symbols is the same, but the runs are delayed because ρ built the output from the left, whereas ρ' did it from the right. This idea of delay comes with an important caveat. As another example, consider two runs ρ and ρ' on the same input that produce the same output *aaaaaa*, and assume that, after consuming the same prefix of the input, ρ and ρ' produced $aaa\_\_\_$ and $\_\_\_aaa$, respectively. Note that the output *aaaaaa* is a periodic word. Hence, it does not matter if *aaa* is appended or prepended to a word with period

a. In general, one copes with this phenomenon by dividing the output into periodic parts, where all periods are bounded by a well-chosen parameter C. So, intuitively, the delay measures the difference between the numbers of output symbols that have been produced by the two runs, up to the end of each of periodic factor. The number of produced output symbols is formally captured by a weight function, defined below, and the delay aggregates the weight differences.

For an accepting run ρ, a position t of ρ, and a position j in the output $\mathrm{out}(\rho)$, we denote by $\mathrm{weight}^t_j(\rho)$ the number of output positions $j' \le j$ that are produced by the prefix of ρ up to position t. We use the above notation when j witnesses a change in a repeating pattern of the output. These changes in repeating patterns are called cuts, as formalized below.

Let w be any non-empty word (e.g. the output of ρ or a factor of it). The *primitive root* of w, denoted $\mathrm{root}(w)$, is the shortest word r such that $w \in \{r\}^*$. For a fixed integer $C > 0$ we define a factorization $w[1, j_1], w[j_1+1, j_2], \dots, w[j_n+1, j_{n+1}]$ of w in which every j_i is chosen as the rightmost position for which $w[j_{i-1}+1, j_i]$ has primitive root of length not exceeding C. These positions $j_1, \dots, j_n$ are called *C-cuts*. More precisely:

- the *first C-cut* of w is the largest position $j \le |w|$, such that $|\mathrm{root}(w[1, j])| \le C$;
- if j is the i-th C-cut of w, then the *$(i+1)$-th C-cut* of w is the largest position $j' > j$ such that $|\mathrm{root}(w[j+1, j'])| \le C$.

We denote by C-$\mathrm{cuts}(w)$ the set of all C-cuts of w.

We are now ready to define the notion of delay. Consider two accepting runs ρ, ρ' of an SST with the *same input* $u = \mathrm{in}(\rho) = \mathrm{in}(\rho')$ and the *same output* $w = \mathrm{out}(\rho) = \mathrm{out}(\rho')$, and define:

$$C\text{-delay}(\rho, \rho') \;=\; \max_{\substack{t \le |u|,\\ j \in C\text{-cuts}(w)}} \left|\mathrm{weight}^t_j(\rho) - \mathrm{weight}^t_j(\rho')\right|$$

In other words, the delay between two such runs ρ and ρ' measures over all input positions the maximal difference between the amount of generated output, but only up to C-cuts of the output. Note that the delay is only defined for accepting runs with same input and output. So whenever we write C-delay(ρ, ρ'), we implicitly mean that ρ, ρ' have *same input and same output*.

EXAMPLE 2.11. Let $w = abcccbb$ be the output of runs ρ, ρ' on the same input of length 2. Assume ρ produces $abc\_\,\_bb$ and then $abcccbb$, whereas ρ' produces $\_\,\_\,\_c\_bb$ and then $abcccbb$. For $C = 2$, we obtain 2-cuts$(w) = \{2, 5, 7\}$, i.e., w is divided into $ab|ccc|bb$. To compute the 2-delay(ρ, ρ'), we need to calculate weights at cuts. For $t = 0$, $\mathrm{weight}^0_j(\rho) = \mathrm{weight}^0_j(\rho') = 0$ for all $j \in$ 2-cuts(w) because nothing has been produced. For $t = 2$, $\mathrm{weight}^2_j(\rho) = \mathrm{weight}^2_j(\rho') = j$ for all $j \in$ 2-cuts(w) because the whole output has been produced. Only the case $t = 1$ has an impact on the delay. We have $\mathrm{weight}^1_2(\rho) = 2$, $\mathrm{weight}^1_5(\rho) = 3$, and $\mathrm{weight}^1_7(\rho) = 5$. Also, we have $\mathrm{weight}^1_2(\rho') = 0$, $\mathrm{weight}^1_5(\rho) = 1$, and $\mathrm{weight}^1_7(\rho) = 3$. Hence, we obtain 2-delay$(\rho, \rho') = 2$. ◆

We recall below some crucial results obtained in [25]. A first result shows that the relation of pairs of runs having bounded delay (for a fixed bound) is automatic — for this to make sense, we view a run of an SST as a finite word, with letters representing transitions, and we recall that a relation is automatic if its convolution language is regular.

LEMMA 2.12 ([25, Theorem 5]). *Given an SST and some numbers C, D, the relation consisting of pairs of accepting runs (ρ, ρ') such that $C\text{-delay}(\rho, \rho') \leq D$ is automatic.*

PROOF. The statement in [25, Theorem 5] is not for runs of SSTs, but for sequences of updates. One can easily build an automaton that checks if two sequences of transitions, encoded by their convolution, form accepting runs ρ, ρ' of the given SST on the same input. The remaining condition $C\text{-delay}(\rho, \rho') \leq D$ only depends on the underlying sequences of updates determined by ρ and ρ', and can be checked using [25, Theorem 5]. ∎

A second result shows that given two runs with large delay, one can find a set of positions on the input (the cardinality of which depends on how large the delay is) in such a way that any interval starting just before any of these positions and ending just after any other of these positions is a loop on both runs such that, when pumped, produces different outputs. Roughly, the reason for obtaining different outputs is that pumping creates a misalignment between C-cuts that were properly aligned before pumping, and different periods cannot overlap. By this last result, large delay intuitively means “potentially different outputs”.

LEMMA 2.13 ([25, Lemma 6]). *Given an SST, one can compute[4] some numbers C, D such that, for all $m \geq 1$ and all runs ρ, ρ': if $Cm\text{-delay}(\rho, \rho') > Dm^2$, then there exist m positions $0 \leq \ell_1 < \cdots < \ell_m \leq |\rho|$ such that, for every $1 \leq i < j \leq m$, the interval $L_{i,j} = [\ell_i, \ell_j]$ is a loop on both ρ and ρ' and satisfies*

$$\text{out}(\text{pump}^2_{L_{i,j}}(\rho)) \neq \text{out}(\text{pump}^2_{L_{i,j}}(\rho')).$$

To reason about finite valuedness we will need to consider *several* accepting runs on the same input, with pairwise large delays. By Lemma 2.13, every two such runs can be pumped so as to witness different outputs. The crux however is to show that these runs can be pumped *simultaneously* so as to get pairwise different outputs. This is indeed possible thanks to:

LEMMA 2.14. *Let C, D be computed as in Lemma 2.13, and k be an arbitrary number. Then one can compute a number m such that, for all runs $\rho_0, \ldots, \rho_k$ on the same input and with $\bigwedge_{0 \leq i < j \leq k} (\text{out}(\rho_i) \neq \text{out}(\rho_j) \ \vee\ Cm\text{-delay}(\rho_i, \rho_j) > Dm^2)$, there is a tuple $\bar{L} = (L_{i,j})_{0 \leq i < j \leq k}$ of disjoint intervals that are loops on all runs $\rho_0, \ldots, \rho_k$, and there is a tuple $\bar{n} = (n_{i,j})_{0 \leq i < j \leq k}$ of positive numbers such that*

$$\textit{for all } 0 \leq i < j \leq k, \quad \text{out}(\text{pump}^{\bar{n}}_{\bar{L}}(\rho_i)) \neq \text{out}(\text{pump}^{\bar{n}}_{\bar{L}}(\rho_j)).$$

4 We remark that the notation and the actual bounds here differ from the original presentation of [25], mainly due to the fact that here we manipulate runs with explicit states and loops with idempotent skeletons. In particular, the parameters C, D, m here correspond respectively to the values $kE^2, \ell E^4, CE^2$ with k, ℓ, C as in [25, Lemma 6], and E as in our Lemma 2.4.

PROOF. We first define m. Let E, H be as in Lemma 2.4, and set $m := m_0 + 1$ for the sequence $m_0, \ldots, m_{k-1}$ defined inductively by $m_{k-1} := k(k+1)$, and $m_h := Em^H_{h+1}$.

We show how to pump the runs in such a way that all pairs of indices $i < j$ witnessing Cm-delay$(\rho_i, \rho_j) > Dm^2$ before pumping, will witness different outputs after pumping. Consider one such pair (i, j), with $i < j$, such that Cm-delay$(\rho_i, \rho_j) > Dm^2$, so in particular, $\mathrm{out}(\rho_i) = \mathrm{out}(\rho_j)$ (if there is no such pair, then all runs have pairwise different outputs, and so we are already done). We apply Lemma 2.13 and obtain a set $I_{i,j,0}$ of $m = m_0 + 1$ positions such that each interval $L = [\ell, \ell']$ with $\ell, \ell' \in I_{i,j,0}$ is a loop on both ρ_i and ρ_j, and:

$$\mathrm{out}(\mathrm{pump}^2_L(\rho_i)) \neq \mathrm{out}(\mathrm{pump}^2_L(\rho_j)). \tag{3}$$

Then, by repeatedly using Lemma 2.4, we derive the existence of sets $I_{i,j,k-1} \subseteq \cdots \subseteq I_{i,j,1} \subseteq I_{i,j,0}$ with $|I_{i,j,h}| = m_h + 1$ such that each interval $L = [\ell, \ell']$ with $\ell, \ell' \in I_{i,j,h}$ is a loop on ρ_i, ρ_j, and h further runs from $\rho_0, \ldots, \rho_k$ (our definition of m from the beginning of the proof is tailored to this repeated application of Lemma 2.4, because $|I_{i,j,h}| = m_h + 1 = Em^H_{h+1} + 1$). In particular, all intervals with endpoints in $I_{i,j,k-1}$ are loops on all the $\rho_0, \ldots, \rho_k$.

In this way, for each pair $i < j$ such that ρ_i and ρ_j have large delay, we obtain $k(k+1)$ adjacent intervals that are loops on all runs and that satisfy the pumping property (3) from above.

As there are at most $k(k+1)$ pairs of runs, we can now choose from the sets of intervals that we have prepared one interval $L_{i,j}$ for each pair $i < j$ with Cm-delay$(\rho_i, \rho_j) > Dm^2$, in such a way that all the chosen intervals are pairwise disjoint (for example, we could do so by always picking among the remaining intervals the one with the left-most right border, and then removing all intervals that intersect this one). The selected intervals $L_{i,j}$ thus have the following properties:

1. $L_{i,j}$ is a loop on all runs $\rho_0, \ldots, \rho_k$,
2. $L_{i,j}$ is disjoint from every other interval $L_{i',j'}$,
3. $\mathrm{out}(\mathrm{pump}^2_{L_{i,j}}(\rho_i)) \neq \mathrm{out}(\mathrm{pump}^2_{L_{i,j}}(\rho_j))$.

If a pair $i < j$ of runs is such that $\mathrm{out}(\rho_i) \neq \mathrm{out}(\rho_j)$, then we set $L_{i,j}$ as an empty loop.

Now, let $\bar{L} = (L_{i,j})_{0 \le i < j \le k}$ be the tuple of chosen intervals, and consider the following system of word inequalities with formal parameters $(\mathrm{x}_{i,j})_{0 \le i < j \le k} =: \bar{\mathrm{x}}$:

$$\text{for all } 0 \le i < j \le k, \quad \mathrm{out}(\mathrm{pump}^{\bar{\mathrm{x}}}_{\bar{L}}(\rho_i)) \neq \mathrm{out}(\mathrm{pump}^{\bar{\mathrm{x}}}_{\bar{L}}(\rho_j))\,.$$

Here, the value of the formal parameter $\mathrm{x}_{i,j}$ determines how often the loop $L_{i,j}$ is pumped. By Corollary 2.6, this corresponds to a word inequality in the parameters $\mathrm{x}_{i,j}$.

Note that there is one such inequality for each pair of runs ρ_i, ρ_j with $0 \le i < j \le k$. By the choice of the intervals in $\bar{L}$, each of the inequalities is satisfiable: indeed, the inequality for ρ_i, ρ_j is satisfied by letting $\mathrm{x}_{i',j'} = 1$ if $i' \neq i$ or $j' \neq j$, and $\mathrm{x}_{i,j} = 2$ otherwise.

By Corollary 2.9, the system of inequalities is also satisfiable with a tuple $\bar{n} = (n_{i,j})_{0 \le i < j \le k}$ of numbers, as claimed in the lemma. ∎

3. The Decomposition Theorem

This section is devoted to the proof of the Decomposition Theorem:

THEOREM 1.2. **(Restated)** *For all $k \in \mathbb{N}$, every k-valued SST can be effectively decomposed into a union of k single-valued (or even deterministic) SSTs. The complexity of the construction is elementary.*

Our proof relies on the notion of *cover* of an SST, which is reminiscent of the so-called "lag-separation covering" construction [21, 46]. Intuitively, given an SST T and two integers $C, D \in \mathbb{N}$, we construct an SST $\mathrm{Cover}_{C,D}(T)$ that is equivalent to T, yet for each input u it only admits pairs of accepting runs with different outputs or C-delay larger than D. The crucial point will be that $\mathrm{Cover}_{C,D}(T)$ is k-ambiguous when T is k-valued.

PROPOSITION 3.1. *Given an SST T and two numbers C, D, one can compute an SST called $\mathrm{Cover}_{C,D}(T)$ such that*

1. *$\mathrm{Cover}_{C,D}(T)$ is equivalent to T;*
2. *for every two accepting runs $\rho \neq \rho'$ of $\mathrm{Cover}_{C,D}(T)$ having the same input, either $\mathrm{out}(\rho) \neq \mathrm{out}(\rho')$ or C-$\mathrm{delay}(\rho, \rho') > D$;*
3. *every accepting run of $\mathrm{Cover}_{C,D}(T)$ can be projected onto an accepting run of T.*

PROOF. We order the set of accepting runs of T lexicographically, and we get rid of all the runs for which there exists a lexicographically smaller run with the same input, the same output, and small delay. Since all these conditions are encoded by regular languages, the remaining set of runs is also regular, and this can be used to construct an SST $\mathrm{Cover}_{C,D}(T)$ that satisfies the required properties.

We now give more details about this construction. Let R denote the set of all accepting runs of T. Remark that R is a language over the alphabet consisting of transitions of T, and it is recognized by the underlying automaton of T, so it is regular. Let

$$\mathrm{Sep}_{C,D}(R) \;=\; \left\{\rho \in R \,\middle|\, \nexists \rho' \in R \,.\, \rho' < \rho \;\wedge\; C\text{-}\mathrm{delay}(\rho, \rho') \le D\right\}.$$

Recall that the delay is only defined for accepting runs with same input and same output, so C-$\mathrm{delay}(\rho, \rho') \le D$ implies that $\mathrm{in}(\rho) = \mathrm{in}(\rho')$ and $\mathrm{out}(\rho) = \mathrm{out}(\rho')$. We show that

A) $\mathrm{Sep}_{C,D}(R)$ is a regular subset of R;

B) $\{(\mathrm{in}(\rho), \mathrm{out}(\rho)) \mid \rho \in \mathrm{Sep}_{C,D}(R)\} = \{(\mathrm{in}(\rho), \mathrm{out}(\rho)) \mid \rho \in R\}$;

C) for every pair of runs $\rho, \rho' \in \mathrm{Sep}_{C,D}(R)$ over the same input, either $\mathrm{out}(\rho) \neq \mathrm{out}(\rho')$ or C-$\mathrm{delay}(\rho, \rho') > D$.

Before proving these properties, let us show how to use them to conclude the proof of the proposition:

We start with a DFA A recognizing $\mathrm{Sep}_{C,D}(R)$, whose existence is guaranteed by Property A). Note that the transitions of A are of the form $\big(q,(s,a,\alpha,s'),q'\big)$, where (s,a,α,s') is a transition of T. Without loss of generality, we assume that the source state q of an A-transition determines the source state s of the corresponding T-transition, and similarly for the target states q' and s'. Thanks to this, we can turn A into the desired SST $\mathrm{Cover}_{C,D}(T)$ by simply projecting away the T-states from the T-transitions, namely, by replacing every transition $\big(q,(s,a,\alpha,s'),q'\big)$ with (q,a,α,q'). To complete the construction, we observe that if the state q' is final in A, then the corresponding state s' is also final in T (this is because A recognizes only accepting runs of T). Accordingly, we can define the final update of $\mathrm{Cover}_{C,D}(T)$ so that it maps any final state q' of A to the final update $O(s')$, as determined by the corresponding final state s' in T. Finally, thanks to Properties B) and C), the SST $\mathrm{Cover}_{C,D}(T)$ constructed in this way clearly satisfies the properties claimed in the proposition.

Let us now prove Properties A)–C).

Proof of Property A). Note that the set $\mathrm{Sep}_{C,D}(R)$ is obtained by combining the relations R, $\{(\rho,\rho') \mid \rho' < \rho\}$, and $\{(\rho,\rho') \mid C\text{-delay}(\rho,\rho') \leq D\}$ using the operations of intersection, projection, and complement. Also recall that R can be regarded a regular language, and that $\{(\rho,\rho') \mid \rho' < \rho\}$ and $\{(\rho,\rho') \mid C\text{-delay}(\rho,\rho') \leq D\}$ are automatic relations (for the latter one uses Lemma 2.12). It is also a standard result (cf. [31, 37, 13]) that automatic relations are closed under intersection, projection, and complement. From this it follows that $\mathrm{Sep}_{C,D}(R)$ is a regular language.

Proof of Property B). As $\mathrm{Sep}_{C,D}(R) \subseteq R$, the left-to-right inclusion is immediate. To prove the converse inclusion, consider an input-output pair (u,v) in the right hand-side of the equation, namely, (u,v) is a pair in the relation realized by T. Let ρ be the lexicographically least accepting run of T such that $\mathrm{in}(\rho) = u$ and $\mathrm{out}(\rho) = v$. By construction, $\rho \in \mathrm{Sep}_{C,D}(R)$ and hence (u,v) also belongs to the left hand-side of the equation.

Proof of Property C). This holds trivially by the definition of $\mathrm{Sep}_{C,D}(R)$. ■

We can now present the missing ingredients of the decomposition result. Proposition 3.2 below shows that, for suitable choices of C and D that depend on the valuedness of T, $\mathrm{Cover}_{C,D}(T)$ turns out to be k-ambiguous.

This will enable the decomposition result via a classical technique that decomposes any k-ambiguous automaton/transducer into a union of k unambiguous ones (see Proposition 3.3 further below).

PROPOSITION 3.2. *Let T be a k-valued SST and let C, D, m be as in Lemma 2.14 (note that m depends on k). The SST* $\mathrm{Cover}_{Cm,Dm^2}(T)$ *is k-ambiguous.*

PROOF. We prove the contrapositive of the statement. Assume that $\mathrm{Cover}_{Cm,Dm^2}(T)$ is not k-ambiguous, that is, it admits $k+1$ accepting runs $\rho_0, \ldots, \rho_k$ on the same input. Recall from Proposition 3.1 that for all $0 \le i < j \le k$, either $\mathrm{out}(\rho_i) \neq \mathrm{out}(\rho_j)$ or Cm-delay$(\rho_i, \rho_j) > Dm^2$. By Lemma 2.14 we can find pumped versions of the runs $\rho_0, \ldots, \rho_k$ that have all the same input but have pairwise different outputs, and thus T is not k-valued. ∎

PROPOSITION 3.3. *For all $k \in \mathbb{N}$, every k-ambiguous SST can be decomposed into a union of k unambiguous SSTs.*

PROOF. The decomposition is done via a classical technique applicable to k-ambiguous NFA and, by extension, to all variants of automata and transducers (see [36, 44]). More precisely, decomposing a k-ambiguous NFA into a union of k unambiguous NFA is done by ordering runs lexicographically and by letting the i-th NFA in the decomposition guess the i-th accepting run on a given input (if it exists). Since the lexicographic order is a regular property of pairs of runs, it is easy to track all smaller runs. ∎

Proof of Theorem 1.2. We now have all the ingredients to prove Theorem 1.2, which directly follows from Propositions 3.2 and 3.3, and the fact that unambiguous SSTs can be determinized [6]. ∎

4. Finite valuedness

We characterize finite valuedness of SSTs by excluding certain types of substructures. Our characterization has strong analogies with the characterization of finite valuedness for one-way transducers, where the excluded substructures have the shape of a "W" and are therefore called *W-patterns* (cf. [21]).[5]

DEFINITION 4.1. A *W-pattern* is a substructure of an SST consisting of states q_1, q_2, r_1, r_2, r_3, and some initial and final states, that are connected by runs as in the diagram of Figure 3.

In that diagram a notation like $\rho : u'/\mu$ describes a run named ρ that consumes an input u' and produces an update μ. Moreover, the cyclic runs ρ''_1, ρ''_2, ρ''_3, $\rho'_1\rho''_1\rho'''_1$, $\rho'_2\rho''_2\rho'''_2$, and $\rho'_3\rho''_3\rho'''_3$ are required to be loops, namely, their updates must have idempotent skeletons.

An important feature of the above definition is that the small loops at states r_1, r_2, r_3 consume the same input, i.e. v'', and, similarly, the big loops at q_1 and q_2, as well as the runs from q_1 to q_2, consume the same set of inputs, i.e. $v'\,(v'')^*\,v'''$.

5 [21] used also other excluded substructures, but they can be seen as degenerate cases of W-patterns.

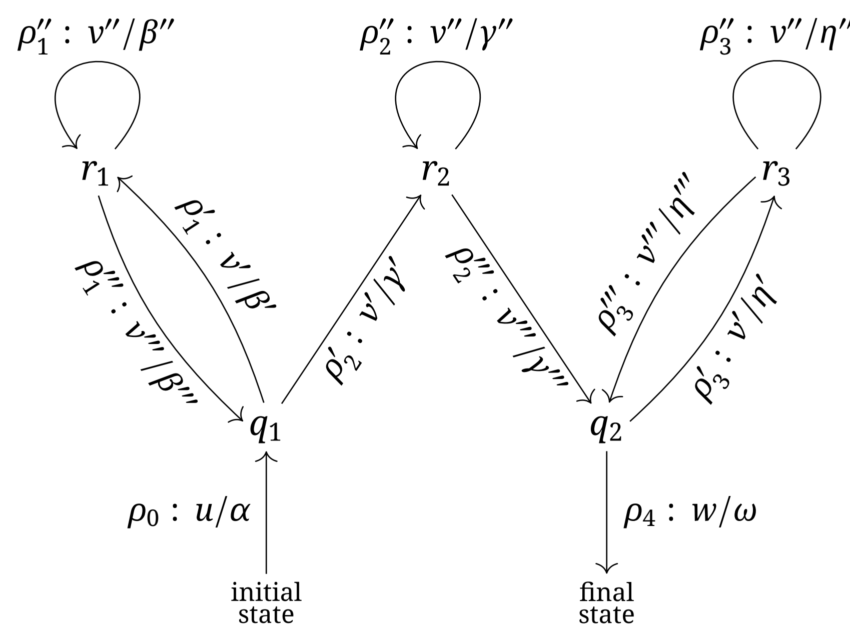


Figure 3. A W-pattern.

Given a W-pattern P and a number $x \in \mathbb{N}_+$, we construct the following runs by pumping the small loops in the diagram of Figure 3 x times:

$$\begin{aligned}
\mathrm{lft}_P^x &= \rho_1' (\rho_1'')^x \, \rho_1''' \\
\mathrm{mid}_P^x &= \rho_2' \, (\rho_2'')^x \, \rho_2''' \\
\mathrm{rgt}_P^x &= \rho_3' \, (\rho_3'')^x \, \rho_3'''.
\end{aligned}$$

Similarly, given a sequence $s = (x_1, x_2, \ldots, x_{i-1}, \underline{x_i}, x_{i+1}, \ldots, x_n)$ of positive numbers with exactly one element underlined (we call such a sequence a *marked sequence*), we define the accepting run

$$\mathrm{run}_P(s) \;=\; \rho_0 \; \underbrace{\mathrm{lft}_P^{x_1} \, \mathrm{lft}_P^{x_2} \, \ldots \, \mathrm{lft}_P^{x_{i-1}}}_{\text{loops at } q_1} \; \mathrm{mid}_P^{x_i} \; \underbrace{\mathrm{rgt}_P^{x_{i+1}} \, \mathrm{rgt}_P^{x_{i+2}} \, \ldots \, \mathrm{rgt}_P^{x_n}}_{\text{loops at } q_2} \; \rho_4.$$

For each marked sequence $s = (x_1, \ldots, x_{i-1}, \underline{x_i}, x_{i+1}, \ldots, x_n)$, $\mathrm{run}_P(s)$ consumes the input

$$u \, v' (v'')^{x_1} v''' \; \ldots \; v' (v'')^{x_n} v''' \, w$$

and produces an output of the form

$$\begin{aligned}
\mathrm{out}(\mathrm{run}_P(s)) \;=\; \big(&\iota \, \alpha \, \beta' (\beta'')^{x_1} \beta''' \; \ldots \; \beta' (\beta'')^{x_{i-1}} \beta''' \\
&\gamma' (\gamma'')^{x_i} \gamma''' \\
&\eta' (\eta'')^{x_{i+1}} \eta''' \; \ldots \; \eta' (\eta'')^{x_n} \eta''' \; \omega \, \omega' \big)(X_1)
\end{aligned}$$

where ι is the initial update and ω' is the final update determined by the final state of $\mathrm{run}_P(s)$. Note that, differently from the output, the input only depends on the *unmarked sequence*, and thus a W-pattern can have accepting runs that consume the same input and produce arbitrarily many different outputs. As an example, consider a W-pattern as in Definition 4.1, where γ''

is the only update that produces output symbols – say γ'' appends letter c to the right of the unique variable. Further suppose that $u = w = \varepsilon$, $v' = v''' = a$, and $v'' = b$. So, on input $(aba)\,(ab^2a)\,\ldots\,(ab^na)$, this W-pattern produces n different outputs: $c, c^2, \ldots, c^n$. The definition and the lemma below generalize this example.

DEFINITION 4.2. A W-pattern P is *divergent* if there is a 5-tuple of numbers $n_1, n_2, n_3, n_4, n_5 \in \mathbb{N}_+$ for which the two runs $\mathrm{run}_P((n_1, n_2, n_3, \underline{n_4}, n_5))$ and $\mathrm{run}_P((n_1, \underline{n_2}, n_3, n_4, n_5))$ produce different outputs (recall that the runs consume the same input). It is called *simply divergent* if in addition $n_1, n_2, n_3, n_4, n_5 \in \{1, 2\}$.

THEOREM 4.3. *An SST is finite-valued iff it does not admit a simply divergent W-pattern.*

The two implications of the theorem are shown in Sections 4.2 and 4.3; effectiveness of the characterization is shown in the next section.

4.1 Effectiveness of finite valuedness

We show in this section a PSPACE decision procedure for the characterization of finite valuedness in terms of absence of simply divergent W-patterns (Theorem 4.3). We also prove that the equivalence problem for deterministic SSTs, known to be in PSPACE, is polynomially reducible to the finite valuedness problem. Despite recent efforts by the community to better understand the complexity of the equivalence problem, it is unknown whether the PSPACE upper bound for equivalence (and hence for finite valuedness) can be improved, as no non-trivial lower bound is known. On the other hand, equivalence (as well as finite valuedness) turns out to be in PTIME when the number of variables is fixed [9].

Our effectiveness procedure uses the following complexity result on the composition of deterministic SSTs, which is of independent interest. It is known that deterministic SSTs are closed under composition because of their equivalence to MSO transductions [6]. An automata-based construction for composition is given in [10]. We show next an exponential construction based on reversible transducers [19].

PROPOSITION 4.4. *Let T_1 and T_2 be two deterministic SSTs realizing the functions $f_1 : \Sigma_1^* \to \Sigma_2^*$ and $f_2 : \Sigma_2^* \to \Sigma_3^*$, respectively Let n_i (resp. m_i) be the number of states (resp. variables) of T_i, and $M = n_1+n_2+m_1+m_2$. One can construct in time exponential in M and polynomial in $|\Sigma_1|+|\Sigma_2|+|\Sigma_3|$ a deterministic SST realizing $f_1 \circ f_2$, with exponentially many states and polynomially many variables in M.*

PROOF. Each T_i can be converted in polynomial time into an equivalent two-way transducer that is *reversible*, i.e., both deterministic and co-deterministic [19]. Reversible two-way transducers can be easily seen to be composable in polynomial time [19], so we obtain a reversible two-way transducer S realizing $f_1 \circ f_2$, with state space polynomial in M. Finally, it suffices to convert S back to a deterministic SST, which can be done in time exponential in the number of states of S and polynomial in the size of the alphabets. This yields a deterministic SST with exponentially many states and polynomially many variables in M (see e.g. [38, 20]). ■

THEOREM 1.5. **(Restated)** *Given any SST T, we can decide in* PSPACE *if T is finite-valued (and if the number of variables is fixed then the complexity is* PTIME*). Moreover, this problem is at least as hard as the equivalence problem for deterministic SSTs.*

PROOF. We start with an overview of the proof. By Theorem 4.3, it suffices to decide whether a given SST T admits a simply divergent W-pattern. Let us fix some tuple $\overline{x} = (x_1, \ldots, x_5) \in \{1, 2\}^5$. We construct an SST $T_{\overline{x}}$ which is *not* single-valued iff T has a W-pattern which is simply divergent for $\overline{x}$. Since checking single-valuedness of SST is decidable [9], we can decide finite valuedness of T by solving single-valuedness problems for all SST $T_{\overline{x}}$, for all tuples $\overline{x} \in \{1, 2\}^5$. Intuitively, we exhibit an encoding of W-patterns P as words u_P, and show that the set of these encodings forms a regular language. The encoding u_P informally consists of the runs that form the W-pattern P, and some of these runs are overlapped to be able to check that they are on the same input. Accordingly, the SST $T_{\overline{x}}$ will take as input such an encoding u_P and produce as outputs the two words $\mathrm{out}(\mathrm{run}_P(s))$ and $\mathrm{out}(\mathrm{run}_P(s'))$, where $s = (x_1, x_2, x_3, \underline{x_4}, x_5)$ and $s' = (x_1, \underline{x_2}, x_3, x_4, x_5)$. To achieve this, $T_{\overline{x}}$ can consume the input u_P while iterating the encoded runs as prescribed by s or s' and simulating the transitions to construct the appropriate outputs. Finally, an analysis of the size of $T_{\overline{x}}$ and of the algorithm from [9] for checking single-valuedness gives the PSPACE upper bound.

Detailed reduction We now explain in detail the reduction to the single-valuedness problem. We will then show how to derive the PSPACE upper bound by inspecting the decidability proof for single-valuedness.

Let $T = (\Sigma, \mathcal{X}, Q, Q_{\mathrm{init}}, Q_{\mathrm{final}}, O, \Delta)$ be the given SST and let $\mathcal{P}$ be the set of all W-patterns of T. We first show that $\mathcal{P}$ is a regular set, modulo some well-chosen encodings of W-patterns as words. Recall that a W-pattern consists of a tuple of runs $P = (\rho_0, \rho'_1, \rho''_1, \rho'''_1, \rho'_2, \rho''_2, \rho'''_2, \rho'_3, \rho''_3, \rho'''_3, \rho_4)$, connected as in the diagram of Definition 4.1. Note that some of those runs share a common input (e.g. $\rho''_1, \rho''_2, \rho''_3$ share the input v''). Therefore, we cannot simply encode P as a sequence of runs $\rho_0, \rho'_1, \ldots$, as otherwise regularity would be lost. Instead, in the encoding we overlap groups of runs over the same input, precisely, the group $\{\rho'_1, \rho'_2, \rho'_3\}$ on input v', the group $\{\rho''_1, \rho''_2, \rho''_3\}$ on input v'', and the group $\{\rho'''_1, \rho'''_2, \rho'''_3\}$ on input v'''. Formally, this is done by taking the convolution of the runs in each group, which results in a word over the alphabet Δ^3. Accordingly, P is encoded as the word

$$u_P \;=\; \rho_0 \,\#\, (\rho'_1 \otimes \rho'_2 \otimes \rho'_3) \,\#\, (\rho''_1 \otimes \rho''_2 \otimes \rho''_3) \,\#\, (\rho'''_1 \otimes \rho'''_2 \otimes \rho'''_3) \,\#\, \rho_4$$

where # is a fresh separator. The language $L_{\mathcal{P}} = \{u_P \mid P \in \mathcal{P}\}$, consisting of all encodings of W-patterns, is easily seen to be regular, recognizable by some automaton $A_{\mathcal{P}}$ which checks that runs forming each convolution share the same input and verify skeleton idempotency (recall that skeletons form a finite monoid). The number of states of the automaton $A_{\mathcal{P}}$ turns out to be polynomial in the number of states of T and in the size of the skeleton monoid, which in turn is exponential in the number of variables.

Next, we construct the SST $T_{\bar{x}}$ as the disjoint union of two deterministic SSTs T_s and $T_{s'}$, where $s = (x_1, x_2, x_3, \underline{x_4}, x_5)$ and $s' = (x_1, \underline{x_2}, x_3, x_4, x_5)$. We only describe T_s, as the construction of $T_{s'}$ is similar. The SST T_s is obtained as a suitable restriction of the composition of two deterministic SSTs T^s_{iter} and T_{exec}, which respectively iterate the runs as prescribed by s and execute the transitions read as input. When fed with the encoding u_P of a W-pattern, T^s_{iter} needs to output $\text{run}_P(s) \in \Delta^*$. More precisely, it takes as input a word of the form $\rho_0 \;\#\; (\rho'_1 \otimes \rho'_2 \otimes \rho'_3) \;\#\; (\rho''_1 \otimes \rho''_2 \otimes \rho''_3) \;\#\; (\rho'''_1 \otimes \rho'''_2 \otimes \rho'''_3) \;\#\; \rho_4$ and produces as output

$$\begin{aligned} \rho_0 \;\; & \rho'_1\,(\rho''_1)^{x_1}\,\rho'''_1 \;\; \rho'_1\,(\rho''_1)^{x_2}\,\rho'''_1 \;\; \rho'_1\,(\rho''_1)^{x_3}\,\rho'''_1 \\ & \rho'_2\,(\rho''_2)^{x_4}\,\rho'''_2 \\ & \rho'_3\,(\rho''_3)^{x_5}\,\rho'''_3 \;\; \rho_4. \end{aligned}$$

The SST T^s_{iter} uses one variable for each non-iterated run (e.g. for ρ_0 and ρ'_1), $x_1 + x_2 + x_3$ variables to store copies of ρ''_1, x_4 variables to store copies of ρ''_2, and x_5 variables to store copies of ρ''_3, and eventually outputs the concatenation of all these variables to obtain $\text{run}_P(s)$. Note that T^s_{iter} does not need to check that the input is a well-formed encoding (this is done later when constructing T_s), so the number of its states and variables is bounded by a constant; on the other hand, the input alphabet, consisting of transitions of T, is polynomial in the size of T.

The construction of T_{exec} is straightforward: it just executes the transitions it reads along the input, thus simulating a run of T. Hence T_{exec} has a single state and the same number of variables as T. Its alphabet is linear in the size of T.

Now, T_s is obtained from the composition $T_{\text{exec}} \circ T^s_{\text{iter}}$ by restricting the input domain to $\mathcal{P}$. It is well-known that deterministic SST are closed under composition and regular domain restriction [6]. By the above constructions, we have

$$T_{\bar{x}}(u_P) \;=\; T_s(u_P) \;\cup\; T_{s'}(u_P) \;=\; \{\text{out}(\text{run}_P(s)), \text{out}(\text{run}_P(s'))\}$$

and hence T contains a W-pattern that is simply divergent for $\bar{x}$ iff $T_{\bar{x}}$ is not single-valued. This already implies the decidability of the existence of a simply divergent W-pattern in T, and hence by Theorem 4.3, of finite valuedness.

Complexity analysis Let us now analyse the complexity in detail. This requires first estimating the size of $T_{\bar{x}}$. Let n_T resp. m_T be the number of states of T, resp. its number of variables. From the previous bounds on the sizes of T_{exec} and T^s_{iter} and Proposition 4.4, we derive that the number of states and variables of $T_{\text{exec}} \circ T^{s}_{\text{iter}}$ is polynomial in both n_T and m_T. Further, restricting the domain to $\mathcal{P}$ is done via a product with the automaton $A_{\mathcal{P}}$, whose size is polynomial in n_T and exponential in m_T. Summing up, the number of states of T_s is exponential in m_T, and polynomial in n_T. Its number of variables is polynomial in both n_T and m_T. And so do $T_{s'}$ and $T_{\overline{x}}$.

As explained in [9], checking single-valuedness of SST reduces to checking non-emptiness of a 1-reversal 2-counter machine of size exponential in the number of variables and polynomial in the number of states. This is fortunate, since it allows us to conclude that checking single-valuedness of $T_{\overline{x}}$ reduces to checking non-emptiness of a 1-reversal 2-counter machine of size just exponential in the number of variables of T. The PSPACE upper bound (and the PTIME upper bound for a fixed number of variables) now follow by recalling that non-emptiness of counter machines with fixed numbers of reversals and counters is in NLOGSPACE [30].

Lower bound For the lower bound, consider two deterministic SST T_1, T_2 over some alphabet Σ with same domain D. Domain equivalence can be tested in PTIME because T_1, T_2 are deterministic. Consider a fresh symbol $\# \notin \Sigma$, and the relation

$$R \;=\; \left\{ \big(u_1\#\ldots\#u_n,\ T_{i_1}(u_1)\#\ldots\#T_{i_n}(u_n)\big) \;\middle|\; {\scriptstyle u_i\in D,\ n\in\mathbb{N}, \atop i_1,\ldots,i_n\in\{1,2\}} \right\}$$

It is easily seen that R is realizable by a (non-deterministic) SST. We claim that R is finite-valued iff it is single-valued, iff T_1 and T_2 are equivalent. If T_1 and T_2 are equivalent, then $T_1(u_j) = T_2(u_j)$ for all $1 \le j \le n$, hence R is single-valued, and so finite-valued. Conversely, if T_1 and T_2 are not equivalent, then $T_1(u) \neq T_2(u)$ for some $u \in D$, and the family of inputs $(u\#)^n u$, with $n \in \mathbb{N}$, witnesses the fact that R is not finite-valued. ■

REMARK 4.5. The proof of the previous theorem can be adapted to show that the equivalence problem for deterministic SSTs and the finite valuedness problem for SSTs are equivalent (modulo polynomial many-one reductions).

As a corollary, we obtain an alternative proof of the following known result:

COROLLARY 1.6 ([53]). (Restated) *Finite valuedness of two-way transducers is decidable in* PSPACE.

PROOF. Observe that a necessary condition for a two-way transducer to be finite-valued is that crossing sequences are bounded. More precisely, if a crossing sequence has a loop then the output of the loop must be empty, otherwise the transducer is not finite valued. Given a bound on the length of crossing sequences the standard conversion into an equivalent SST applies, see e.g. [38, 20]. This yields an SST with an exponential number of states and a linear number of variables, both in the number of states of the initial two-way transducer. Finally, we apply the algorithm of Theorem 1.5, and we observe that it amounts to checking emptiness of a 1-reversal 2-counter machine whose number of states is exponential in the number of states of the initial two-way transducer. We conclude again by applying the NLogSpace algorithm for checking emptiness of such counter machines [30]. ■

4.2 A necessary condition for finite valuedness

Here we prove the contrapositive of the left-to-right implication of Theorem 4.3: we show that a divergent W-pattern can generate arbitrarily many outputs on the same input.

LEMMA 4.6. *Every SST that contains some divergent W-pattern is not finite-valued.*

PROOF. Let us fix an SST with a divergent W-pattern P. In order to prove that the SST is not finite-valued, we show that we can construct arbitrary many accepting runs of P that consume the same input and produce pairwise different outputs. To do this we will consider for some suitable $M \in \mathbb{N}$ inequalities in the formal parameters $\mathsf{s}_1, \dots, \mathsf{s}_M \in \mathbb{N}_+$ (where $1 \leq i < j \leq M$), and look for arbitrary large, satisfiable, sets of inequalities of the form:

$$e_{M,i,j}[\mathsf{s}_1, \mathsf{s}_2, \dots, \mathsf{s}_M] : \quad \begin{array}{c} \mathrm{out}(\mathrm{run}_P(\mathsf{s}_1, \mathsf{s}_2, \dots, \mathsf{s}_{i-1}, \underline{\mathsf{s}_i}, \mathsf{s}_{i+1}, \dots, \mathsf{s}_M)) \\ \neq \\ \mathrm{out}(\mathrm{run}_P(\mathsf{s}_1, \mathsf{s}_2, \dots, \mathsf{s}_{j-1}, \underline{\mathsf{s}_j}, \mathsf{s}_{j+1}, \dots, \mathsf{s}_M)). \end{array}$$

Recall that, according to the diagram of Definition 4.1, the number of variable occurrences before (resp. after) the underlined parameter represents the number of loops at state q_1 (resp. q_2) in a run of the W-pattern. Moreover, each variable s_i before (resp. after) the underlined parameter represents the number of repetitions of the small loops at r_1 (resp. r_3) within occurrences of bigger loops at q_1 (resp. q_2); similarly, the underlined variable represents the number of repetitions of the loop at r_2 within the run that connects q_1 to q_2. In view of this, by Corollary 2.6, the outputs of the runs considered in the above inequality have the format required for a word inequality with repetitions parametrized by $\mathsf{s}_1, \dots, \mathsf{s}_M$.

The fact that the W-pattern P is divergent will help to find sets of satisfiable inequalities $e_{M,i,j}$ of arbitrary large cardinality. This, in turn, will produce (combined with our word combinatorics results) arbitrary many accepting runs over the same input, having pairwise different outputs.

CLAIM. *For every $m \in \mathbb{N}$, there exist $M \in \mathbb{N}$ and a set $I \subseteq \{1, 2, \dots, M\}$ of cardinality $m + 1$ such that, for all $i < j \in I$, $e_{M,i,j}$ is satisfiable.*

PROOF OF THE CLAIM. Since P is a divergent W-pattern, there exist $n_1, n_2, n_3, n_4, n_5 \in \mathbb{N}_+$ such that

$$\mathrm{out}(\mathrm{run}_P(n_1, \underline{n_2}, n_3, n_4, n_5)) \neq \mathrm{out}(\mathrm{run}_P(n_1, n_2, n_3, \underline{n_4}, n_5)).$$

We fix such numbers $n_1, n_2, n_3, n_4, n_5 \in \mathbb{N}_+$. Consider now the following inequality over the formal parameters x, y, z:

$$e[\mathrm{x},\mathrm{y},\mathrm{z}] : \quad \begin{array}{c} \mathrm{out}(\mathrm{run}_P(\overbrace{n_1, n_1, \ldots, n_1}^{\text{x times}},\ \underline{n_2},\ \overbrace{n_3, \ldots, n_3}^{\text{y times}},\ n_4,\ \overbrace{n_5, \ldots, n_5}^{\text{z times}})) \\ \neq \\ \mathrm{out}(\mathrm{run}_P(\underbrace{n_1, n_1, \ldots, n_1}_{\text{x times}},\ n_2,\ \underbrace{n_3, \ldots, n_3}_{\text{y times}},\ \underline{n_4},\ \underbrace{n_5, \ldots, n_5}_{\text{z times}})). \end{array}$$

Note that every instance of $e[\mathrm{x},\mathrm{y},\mathrm{z}]$ with concrete values x, y, z is also an instance of $e_{M,i,j}$, where $M = x + y + z + 2$, $i = x + 1$, $j = x + y + 2$, and all parameters $\mathrm{s}_1, \ldots, \mathrm{s}_M$ are instantiated with values from $\{n_1, \ldots, n_5\}$. Moreover, as the parameters in $e[\mathrm{x},\mathrm{y},\mathrm{z}]$ determine the number of repetitions of n_1, n_3, n_5, which in their turn correspond to pumping loops at q_1 and q_2, by Corollary 2.6, the outputs of the considered runs have the format required for a word inequality with repetitions parametrized by x, y, z.

Since $e[\mathrm{x},\mathrm{y},\mathrm{z}]$ is satisfiable (e.g. with $\mathrm{x} = \mathrm{y} = \mathrm{z} = 1$), Corollary 2.10 implies that

$$\exists \ell_\mathrm{y}\ \forall h_\mathrm{y}\ \exists \ell_\mathrm{x}\ \forall h_\mathrm{x}\ \exists \ell_\mathrm{z}\ \forall h_\mathrm{z} \quad \underbrace{[\ell_\mathrm{x}, h_\mathrm{x}]}_{\text{values for x}} \times \underbrace{[\ell_\mathrm{y}, h_\mathrm{y}]}_{\text{values for y}} \times \underbrace{[\ell_\mathrm{z}, h_\mathrm{z}]}_{\text{values for z}} \ \subseteq\ \mathrm{Sols}(e).$$

Note that we start by quantifying over ℓ_y and not ℓ_x (Corollary 2.10 is invariant with respect to the parameter order). exist three integers $\ell_y, \ell_x, \ell_z > 0$ such that

$$[\ell_\mathrm{x}, \ell_\mathrm{x} + 2m\ell_\mathrm{y}] \times [\ell_\mathrm{y}, 2m\ell_\mathrm{y}] \times [\ell_\mathrm{z}, \ell_\mathrm{z} + 2m\ell_\mathrm{y}] \ \subseteq\ \mathrm{Sols}(e). \tag{4}$$

Note that $h_y = 2m\ell_y$ depends only on ℓ_y, while $h_x = \ell_x + 2m\ell_y$ depends on both ℓ_x and ℓ_y.

We can now prove the claim by letting $M = \ell_\mathrm{x} + 2m\ell_\mathrm{y} + \ell_\mathrm{z} + 1$ and $I = \{\ell_\mathrm{x} + 2\lambda\ell_\mathrm{y} + 1 \mid 0 \leq \lambda \leq m\}$. The gap between two consecutive values of I equals $2\ell_y$, and for every $i < j \in I$ we get

$$\begin{array}{llll} i - 1 & \in\ [\ell_\mathrm{x}, \ell_\mathrm{x} + 2(m-1)\ell_\mathrm{y}] & \subseteq & [\ell_\mathrm{x}, \ell_\mathrm{x} + 2m\ell_\mathrm{y}] \\ j - i - 1 & \in\ [2\ell_\mathrm{y} - 1, 2m\ell_\mathrm{y} - 1] & \subseteq & [\ell_\mathrm{y}, 2m\ell_\mathrm{y}] \\ M - j & \in\ [\ell_\mathrm{z}, \ell_\mathrm{z} + 2(m-1)\ell_\mathrm{y}] & \subseteq & [\ell_\mathrm{z}, \ell_\mathrm{z} + 2m\ell_\mathrm{y}]. \end{array}$$

Thus, by Equation (4), $(i-1, j-i-1, M-j) \in \mathrm{Sols}(e)$. This solution of e corresponds to the instance of $e_{M,i,j}$ with the values for the formal parameters $\mathrm{s}_1, \ldots, \mathrm{s}_M$ defined by

$$\mathrm{s}_h = \begin{cases} n_1 & \text{for every } 1 \leq h \leq i-1, \\ n_2 & \text{for } h = i, \\ n_3 & \text{for every } i+1 \leq h \leq j-1, \\ n_4 & \text{for } h = j, \\ n_5 & \text{for every } j+1 \leq M. \end{cases}$$

Hence, $e_{M,i,j}$ is satisfiable for all $i < j \in I$, as claimed. ■

We can now conclude the proof of the lemma using the above claim: Corollary 2.9 tells us that any system of word inequalities is satisfiable when every word inequality in it is so. Using this and the above claim, we derive that for every m there exist $t_1, t_2, \ldots, t_M \in \mathbb{N}_+$ such that, for all $i < j \in I$ (with I as in the claim), $e_{M,i,j}[t_1, t_2, \ldots, t_M]$ holds. For every $h \in I$, let

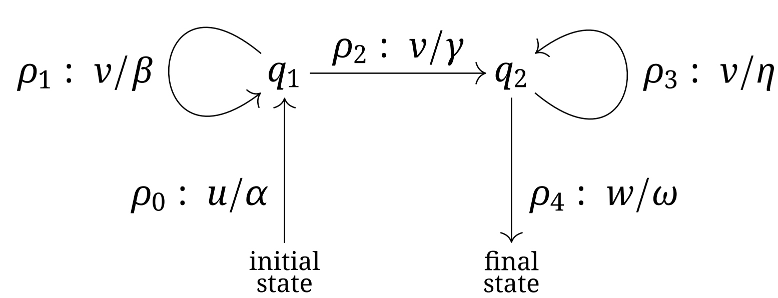
$$\rho_h \;=\; \mathrm{run}_P(t_1, t_2, \ldots, t_{h-1}, \underline{t_h}, t_{h+1}, \ldots, t_M).$$

Note that all runs ρ_h, for $h \in I$, consume the same input, since they all correspond to the same unmarked sequence $(t_1, t_2, \ldots, t_M)$. However, they produce pairwise different outputs, because for every $i < j \in I$, the tuple $(t_1, t_2, \ldots, t_M)$ is a solution of $e_{M,i,j}$. Since $|I| = m$ can be chosen arbitrarily, the transducer is not finite-valued. ■

4.3 A sufficient condition for finite valuedness

We finally prove that any SST that exhibits no simply divergent W-pattern is finite-valued. The proof relies on two crucial results. The first one is a characterization of finite ambiguity for SSTs, which is easily derived from the characterization of finite ambiguity for finite state automata [39, 34, 52, 4]:

DEFINITION 4.7. A *dumbbell* is a substructure of an SST consisting of states q_1, q_2 connected by runs as in the diagram



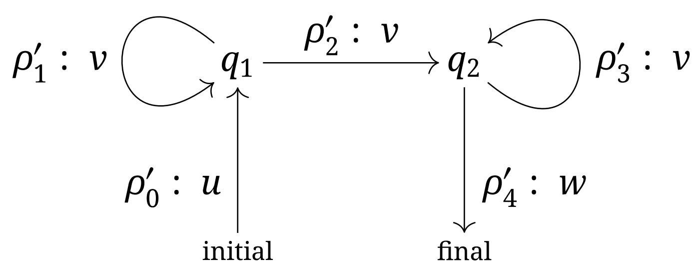
where the runs ρ_1 and ρ_3 are loops (in particular, they produce updates with idempotent skeletons) and at least two among the runs ρ_1, ρ_2, ρ_3 are distinct.

LEMMA 4.8. *An SST is finite-ambiguous iff it does not contain any dumbbell.*

PROOF. Let $T = (\Sigma, \mathcal{X}, Q, Q_{\mathrm{init}}, Q_{\mathrm{final}}, O, \Delta)$ be an SST. By projecting away the updates on the transitions we obtain from T a *multiset finite-state automaton* A. Formally, $A = (\Sigma, Q, Q_{\mathrm{init}}, Q_{\mathrm{final}}, \Delta')$, where Δ' is the *multiset* containing one occurrence of a triple (q, a, q') for each transition of the form (q, a, α, q') in Δ. Note that a multiset automaton can admit several occurrences of the same (accepting) run. Accordingly, the notion of finite ambiguity for A requires the existence of a uniform bound on the number of *occurrences* of accepting runs of A on the same input. We also remark that multiset automata are essentially the same as weighted automata over the semiring of natural numbers (the weight of a transition being its number of occurrences), with only a difference in terminology where finite ambiguity in multiset automata corresponds to finite valuedness in weighted automata.

Given the above construction of A from T, one can verify by induction on $|u|$ that the number of occurrences of accepting runs of A on u coincides with the number of accepting runs of T on u. This means that A is finite-ambiguous iff T is finite-ambiguous.

Finally, we recall the characterizations of finite ambiguity from [39, 34, 52] (see in particular Theorem 1.1 and Lemma 2.6 from [39]). In short, their results directly imply that a multiset automaton is finite-ambiguous iff it does not contain a *plain dumbbell*, namely, a substructure of the form



where at least two among $\rho'_1, \rho'_2, \rho'_3$ are distinct runs.

This almost concludes the proof of the lemma, since any dumbbell of T can be projected into a plain dumbbell of A. The converse implication, however, is not completely straightforward. The reason is that the cyclic runs of a plain dumbbell in A do not necessarily correspond to loops in the SST T, as the runs need not produce updates with idempotent skeletons. Nonetheless, we can reason as follows. Suppose that A contains a plain dumbbell, with occurrences of runs $\rho'_0, \rho'_1, \rho'_2, \rho'_3, \rho'_4$ as depicted above. Let $\rho_0, \rho_1, \rho_2, \rho_3, \rho_4$ be some corresponding runs in T (with ρ_i projecting to ρ'_i) and let $\alpha, \beta, \gamma, \eta, \omega$ be their induced updates. Further let n be a large enough number such that β^n and η^n have idempotent skeletons (such an n always exists since the skeleton monoid is finite). Now consider the substructure in T given by the runs ρ_0, $(\rho_1)^n$, $(\rho_1)^{n-1}\,\rho_2$, $(\rho_3)^n$, and ρ_4. This substructure satisfies precisely the definition of dumbbell for the SST T. ∎

The second ingredient for the proof of the right-to-left implication of Theorem 4.3 uses once more the cover construction described in Proposition 3.1. More precisely, in Lemma 4.9 below we show that if an SST T has no simply divergent W-pattern, then, for some well chosen values C, D, m, the SST $\mathrm{Cover}_{Cm,Dm^2}(T)$ contains no dumbbell. Before proving the lemma, let us show how it can be used to establish the right-to-left implication of Theorem 4.3.

Proof of Theorem 4.3 By Lemma 4.8, if $\mathrm{Cover}_{Cm,Dm^2}(T)$ has no dumbbell, then it is finite-ambiguous, hence finite-valued. Since $\mathrm{Cover}_{Cm,Dm^2}(T)$ and T are equivalent, T is finite-valued, too. ∎

LEMMA 4.9. *Given an SST T, one can compute numbers C, D, m such that if* $\mathrm{Cover}_{Cm,Dm^2}(T)$ *contains a dumbbell, then T contains a simply divergent W-pattern.*

PROOF. We first provide some intuition. If $\mathrm{Cover}_{Cm,Dm^2}(T)$ contains a dumbbell for suitable values of C, D, m, then we show that this dumbbell admits two distinct runs π, π' that have either different outputs or large delay. In both cases we will be able to transform the dumbbell into a simply divergent W-pattern in $\mathrm{Cover}_{Cm,Dm^2}(T)$, hence T will have one as well.

Formally, let T be an SST. Let C, D be defined as in Lemma 2.13, and $m = 7E^{H^2+H+1} + 1$, where E, H are defined as in Lemma 2.4. Next, suppose that $\mathrm{Cover}_{Cm,Dm^2}(T)$ contains a dumbbell as in Definition 4.7, with runs $\rho_0, \rho_1, \rho_2, \rho_3, \rho_4$ that produce respectively the updates $\alpha, \beta, \gamma, \eta, \omega$.

Consider the following accepting runs, which are obtained by composing the copies of the original runs of the dumbbell. The runs π, π' are different because at least two of ρ_1, ρ_2, ρ_3 are different:

$$\begin{aligned} \pi &= \rho_0\ \rho_1\ \rho_2\ \rho_3\ \rho_3\ \rho_3\ \rho_4 \\ \pi' &= \rho_0\ \rho_1\ \rho_1\ \rho_1\ \rho_2\ \rho_3\ \rho_4. \end{aligned} \tag{5}$$

By the properties of $\mathrm{Cover}_{Cm,Dm^2}(T)$, since π and π' consume the same input, they either produce different outputs or have Cm-delay larger than Dm^2.

We first consider the case where the outputs are different. In this case, we can immediately witness a simply divergent W-pattern P by adding empty runs to the dumbbell; formally, for every $i = 1, 2, 3$, we let $\rho'_i = \rho_i$ and $\rho''_i = \rho'''_i = \varepsilon$, so as to form a W-pattern like the one in Figure 3, with $r_1 = q_1$ and $r_2 = r_3 = q_2$. Using the notation introduced at the beginning of Section 4, we observe that $\pi = \mathrm{run}_P(1, \underline{1}, 1, 1, 1)$ and $\pi' = \mathrm{run}_P(1, 1, 1, \underline{1}, 1)$ — recall that the underlined number represents how many times the small loop at r_2, which is empty here, is repeated along the run from q_1 to q_2, and the other numbers represent how many times the small loops at r_1 and r_3, which are also empty here, are repeated within the occurrences of big loops at q_1 and q_2. Since, by assumption, the runs π and π' produce different outputs, the W-pattern P is simply divergent, as required.

We now consider the case where π and π' have large delay, namely, $Cm\text{-delay}(\pi, \pi') > Dm^2$. In this case Lemma 2.13 guarantees the existence of a set $I \subseteq \{0, 1, \ldots, |\pi|\}$ containing m positions in between the input letters such that, for all pairs $i < j$ in I, the interval $[i, j]$ is a loop on both π and π' and satisfies

$$\mathrm{out}(\mathrm{pump}^2_{[i,j]}(\pi)) \neq \mathrm{out}(\mathrm{pump}^2_{[i,j]}(\pi')). \tag{6}$$

Next, recall from Equation (5) that π, and similarly π', consists of seven parts, representing copies of the original runs of the dumbbell and consuming the inputs u, v, v, v, v, v, w. We identify these parts with the numbers $1, \ldots, 7$. Since we defined m as $7E^{H^2+H+1} + 1$, there is one of these parts in which at least $E^{H^2+H+1}+1$ of the aforementioned positions of I occur. Let $p \in \{1, 2, \ldots, 7\}$ denote the number of this part, and let I_p be a set of $E^{H^2+H+1} + 1$ positions from I that occur entirely inside the p-th part. We conclude the proof by a further case distinction, depending on whether $p \in \{1, 7\}$ or $p \in \{2, \ldots, 6\}$.

Parts 1 and 7 Let us suppose that $p \in \{1, 7\}$, and let i and j be two distinct positions in I_p. We let P be the W-pattern obtained by transforming the dumbbell as follows:

1. First, we pump either ρ_0 or ρ_4 depending on p:
   - If $p = 1$, we set $\rho'_0 = \mathrm{pump}^2_{[i,j]}(\rho_0)$ and $\rho'_4 = \rho_4$.
   - If $p = 7$, we set $\rho'_4 = \mathrm{pump}^2_{[i-|u\,v^5|, j-|u\,v^5|]}(\rho_4)$ and $\rho'_0 = \rho_0$.
2. Then, we add empty runs to ρ_1, ρ_2, and ρ_3; formally, for each $h \in \{1, 2, 3\}$, we set $\rho'_h = \rho_h$ and $\rho''_h = \rho'''_h = \varepsilon$.

Now that we identified a W-pattern P in $\mathrm{Cover}_{C,D}(T)$, we note that

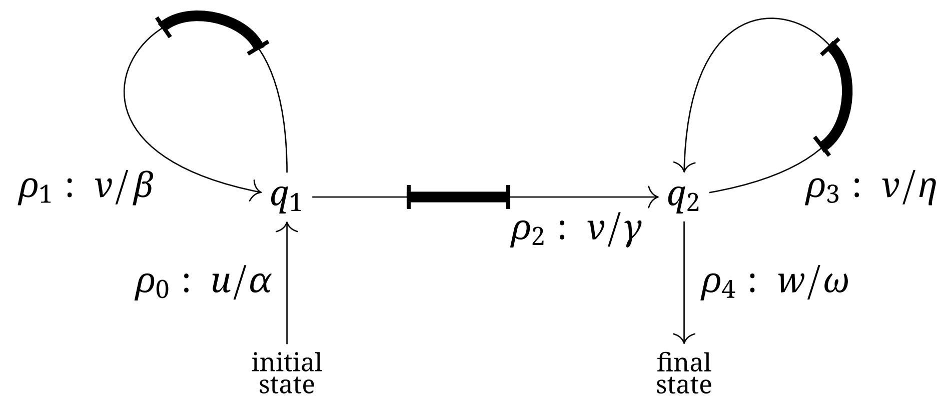
$$\begin{aligned}\mathrm{pump}^2_{[i,j]}(\pi) &= \mathrm{run}_P(1, \underline{1}, 1, 1, 1)\\ \mathrm{pump}^2_{[i,j]}(\pi') &= \mathrm{run}_P(1, 1, 1, \underline{1}, 1).\end{aligned}$$

We also recall Equation 6, which states that these runs produce different outputs. This means that the W-pattern P is simply divergent. Finally, since the runs of $\mathrm{Cover}_{C,D}(T)$ can be projected into runs of T, we conclude that T contains a simply divergent W-pattern.

Parts 2 – 6 Let us suppose that $p \in \{2, \ldots, 6\}$. Note that, in this case, the elements of I_p denote positions inside the p-th factor of the input $u\,v\,v\,v\,v\,v\,w$, which is a v. To refer directly to the positions of v, we define I'_p as the set obtained by subtracting $|u\,v^{p-1}|$ from each element of I_p. Since the set $|I'_p|$ has cardinality $E^{H^2+H+1} + 1$, we claim that we can find an interval with endpoints from I'_p that is a loop of ρ_1, ρ_2, and ρ_3, at the same time. Specifically, we can do so via three consecutive applications of Lemma 2.4:

1. As $|I'_p| = E^{H^2+H+1} + 1 = E \cdot (E^{H+1})^H + 1$, there exists a set $I''_p \subseteq I'_p$ of cardinality $E^{H+1} + 1$ such that for every pair $i < j$ in I''_p, the interval $[i, j]$ is a loop of ρ_1;
2. As $|I''_p| = E^{H+1} + 1 = E \cdot E^H + 1$, there exists $I'''_p \subseteq I''_p$ of cardinality $E + 1$ s.t. for every pair $i < j$ in I'''_p, the interval $[i, j]$ is a loop of ρ_2 (and also of ρ_1, since $i, j \in I'''_p \subseteq I''_p$);
3. As $|I'''_p| = E + 1 = E \cdot 1^H + 1$, there are two positions $i < j$ in I'''_p such that the interval $[i, j]$ is a loop of ρ_3 (and also of ρ_1 and ρ_2 since $i, j \in I'''_p \subseteq I''_p$).

The diagram below summarizes the current situation: we have just managed to find an interval $[i, j]$ that is a loop on all v-labelled runs ρ_1, ρ_2, ρ_3 of the dumbbell (the occurrences of this interval inside ρ_1, ρ_2, ρ_3 are highlighted by thick segments):



We can now expose a W-pattern P by merging the positions i and j inside each v-labelled run ρ_1, ρ_2, and ρ_3 of the dumbbell. Formally, we let $\rho'_0 = \rho_0$, $\rho'_4 = \rho_4$, and for every $h \in \{1,2,3\}$, we define ρ'_h, ρ''_h, and ρ'''_h, respectively, as the intervals $[0,i]$, $[i,j]$, and $[j,|v|]$ of ρ_h. Now that we have identified a W-pattern P inside $\mathrm{Cover}_{C,D}(T)$, we remark that $\pi = \mathrm{run}_P(1,\underline{1},1,1,1)$ and $\pi' = \mathrm{run}_P(1,1,1,\underline{1},1)$. Additionally, if we transpose i and j from I'_p back to I, that is, if we set $i' = i + |u\,v^{p-1}|$ and $j' = j + |u\,v^{p-1}|$, since both i' and j' occur in the p-th part of π and π', pumping the interval $[i',j']$ in π (resp. π') amounts to incrementing the $(p-1)$-th parameter in the notation $\mathrm{run}_P(1,\underline{1},1,1,1)$ (resp. $\mathrm{run}_P(1,1,1,\underline{1},1)$). More precisely:

$$\begin{aligned}\mathrm{pump}^2_{[i',j']}(\pi) &= \mathrm{run}_P(n_1,\underline{n_2},n_3,n_4,n_5)\\ \mathrm{pump}^2_{[i',j']}(\pi') &= \mathrm{run}_P(n_1,n_2,n_3,\underline{n_4},n_5)\end{aligned}$$

where each $n_{p'}$ is either 2 or 1 depending on whether $p' = p-1$ or not. Since Equation 6 states that these two runs produce different outputs, the W-pattern P is simply divergent. Finally, since the runs of $\mathrm{Cover}_{C,D}(T)$ can be projected into runs of T, we conclude that, also in this case, T contains a simply divergent W-pattern. ∎

5. Conclusion

We have drawn a rather complete picture of finite-valued SSTs and answered several open questions of [9]. Regarding expressiveness, finite-valued SSTs can be decomposed as unions of deterministic SSTs (Theorem 1.2). They are equivalent to finite-valued two-way transducers (Theorem 1.3), and to finite-valued non-deterministic MSO transductions (see Section 1). On the algorithmic side, their equivalence problem is decidable in elementary time (Theorem 1.4) and finite valuedness of SSTs is decidable in PSpace (PTime for fixed number of variables), see Theorem 1.5. As an alternative proof to the result of [53], our results imply that finite valuedness of two-way transducers can be decided in PSpace (Corollary 1.6). Because of the effective expressiveness equivalence between SSTs and non-deterministic MSO transductions, our result also entails decidability of finite valuedness for the latter class.

Further questions. A first natural question is how big the valuedness of an SST can be. In the classical case of one-way transducers the valuedness has been shown to be at most exponential (if finite) [51]. We can obtain a bound from Lemma 4.9, but the value is likely to be sub-optimal.

Our equivalence procedure relies on the decomposition of a k-valued SST into a union of k deterministic SSTs each of elementary size. The latter construction is likely to be sub-optimal, too, and so is our complexity for checking equivalence. On the other hand, only a PSPACE lower bound is known, which follows easily from a reduction of NFA equivalence [1]. A better understanding of the complexity of the equivalence problem for (sub)classes of SSTs is a challenging question. Already for deterministic SSTs, the complexity of the equivalence problem is only known to lie between NLOGSPACE and PSPACE [7].

However, beyond the finite-valued setting there is little hope to find a natural restriction on valuedness which would preserve the decidability of the equivalence problem. Already for one-way transducers of linear valuedness (i.e. where the number of outputs is linear in the input length), equivalence is undecidable, as shown through a small modification of the proof of [32].

Deterministic SSTs have been extended, while preserving decidability of the equivalence problem, in several ways: to copyful SSTs [15, 26], which allow to copy the content of variables several times, to infinite strings [11], and to trees [8]. Generalizations of these results to the finite-valued setting yield interesting questions. On trees, similar questions (effective finite valuedness, decomposition and equivalence) have been answered positively for bottom-up tree transducers [47].

Finally, SSTs have linear input-to-output growth (in the length of the strings). There is a recent trend in extending transducer models to allow polynomial growth [17, 16, 14, 22]. Finite valuedness has not been studied in this context yet.

References

[1] **Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman**. The Design and Analysis of Computer Algorithms. Addison-Wesley, New York, 1974. (34)

[2] **Michael H. Albert and John Lawrence**. A proof of Ehrenfeucht's Conjecture. *Theor. Comput. Sci.* 41:121–123, 1985. DOI (4)

[3] **Cyril Allauzen and Mehryar Mohri**. Efficient algorithms for testing the twins property. *J. Autom. Lang. Comb.* 8(2):117–144, 2003. DOI (3)

[4] **Cyril Allauzen, Mehryar Mohri, and Ashish Rastogi**. General algorithms for testing the ambiguity of finite automata and the double-tape ambiguity of finite-state transducers. *Int. J. Found. Comput. Sci.* 22(4):883–904, 2011. DOI (29)

[5] **Rajeev Alur, Mikołaj Bojańczyk, Emmanuel Filiot, Anca Muscholl, and Sarah Winter**. Regular Transformations (Dagstuhl Seminar 23202). *Dagstuhl Reports*, 13(5):96–113, 2023. DOI (1)

[6] **Rajeev Alur and Pavol Cerný**. Expressiveness of streaming string transducers. *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. DOI (2, 5, 6, 21, 23, 25)

[7] **Rajeev Alur and Pavol Cerný**. Streaming transducers for algorithmic verification of single-pass list-processing programs. *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 599–610. ACM, 2011. DOI (8, 34)

[8] **Rajeev Alur and Loris D'Antoni**. Streaming tree transducers. *J. ACM*, 64(5):31:1–31:55, 2017. DOI (34)

[9] **Rajeev Alur and Jyotirmoy V. Deshmukh**. Nondeterministic streaming string transducers. *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011. DOI (3–6, 8, 23, 24, 26, 33)

[10] **Rajeev Alur, Taylor Dohmen, and Ashutosh Trivedi**. Composing copyless streaming string transducers. *CoRR*, abs/2209.05448:1–21, 2022. DOI (23)

[11] **Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi**. Regular transformations of infinite strings. *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 65–74. IEEE Computer Society, 2012. DOI (34)

[12] **Marie-Pierre Béal and Olivier Carton**. Determinization of transducers over finite and infinite words. *Theor. Comput. Sci.* 289(1):225–251, 2002. DOI (3)

[13] **Achim Blumensath and Erich Grädel**. Automatic structures. *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 51–62. IEEE Computer Society, 2000. DOI (6, 20)

[14] **Mikołaj Bojańczyk**. On the growth rates of polyregular functions. *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*, pages 1–13. IEEE, 2023. DOI (34)

[15] **Mikołaj Bojańczyk**. The Hilbert method for transducer equivalence. *ACM SIGLOG News*, 6(1):5–17, 2019. DOI (34)

[16] **Mikołaj Bojańczyk**. Transducers of polynomial growth. *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, 1:1–1:27. ACM, 2022. DOI (34)

[17] **Mikołaj Bojańczyk, Sandra Kiefer, and Nathan Lhote**. String-to-string interpretations with polynomial-size output. *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, 106:1–106:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI (34)

[18] **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. DOI (2)

[19] **Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote**. On reversible transducers. *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, 113:1–113:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. DOI (23)

[20] **Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier**. Aperiodic string transducers. *Int. J. Found. Comput. Sci.* 29(5):801–824, 2018. DOI (5, 24, 26)

[21] **Rodrigo De Souza**. Etude structurelle des transducteurs de norme bornée. PhD thesis, LTCI - Laboratoire Traitement et Communication de l'Information, Paris-Saclay, 2008. URL (19, 21)

[22] **Gaëtan Douéneau-Tabot**. Optimization of string transducers. PhD thesis, Université Paris Cité, Paris, France, 2023. URL (10, 34)

[23] **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 (2)

[24] **Emmanuel Filiot, Ismaël Jecker, Christof Löding, Anca Muscholl, Gabriele Puppis, and Sarah Winter**. Finite-valued streaming string transducers. *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, 33:1–33:14. ACM, 2024. DOI (1)

[25] **Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter**. A regular and complete notion of delay for streaming string transducers. *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, 32:1–32:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. DOI (6, 15–17)

[26] **Emmanuel Filiot and Pierre-Alain Reynier**. Copyful streaming string transducers. *Fundam. Informaticae*, 178(1-2):59–76, 2021. DOI (34)

[27] **Patrick C. Fischer and Arnold L. Rosenberg**. Multitape one-way nonwriting automata. *J. Comput. Syst. Sci.* 2(1):88–101, 1968. DOI (3, 8)

[28] **Victor S. Guba**. Equivalence of infinite systems of equations in free groups and semigroups to finite subsystems. *Mat. Zametki*, 40(3):688–690, 1986. DOI (4)

[29] **Eitan M. Gurari and Oscar H. Ibarra**. A note on finitely-valued and finitely ambiguous transducers. *Math. Syst. Theory*, 16(1):61–66, 1983. DOI (3)

[30] **Eitan M. Gurari and Oscar H. Ibarra**. The complexity of decision problems for finite-turn multicounter machines. *J. Comput. Syst. Sci.* 22(2):220–229, 1981. DOI (8, 26, 27)

[31] **Bernard R. Hodgson**. Décidabilité par automate fini. *Ann. Sci. Math. Québec*, 7(3):39–57, 1983. URL (20)

[32] **Oscar H. Ibarra**. The unsolvability of the equivalence problem for epsilon-free NGSM's with unary input (output) alphabet and applications. *SIAM J. Comput.* 7(4):524–532, 1978. DOI (3, 8, 34)

[33] **Karel Culík II and Juhani Karhumäki**. The equivalence of finite valued transducers (on HDT0L languages) is decidable. *Theor. Comput. Sci.* 47(3):71–84, 1986. DOI (4)

[34] **Gérard Jacob**. Un algorithme calculant le cardinal, fini ou infini, des demi-groupes de matrices. *Theor. Comput. Sci.* 5(2):183–204, 1977. DOI (29, 30)

[35] **Ismaël Jecker**. A Ramsey theorem for finite monoids. *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, 44:1–44:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. DOI (9)

[36] **J. Howard Johnson**. Do rational equivalence relations have regular cross-sections? *Automata, Languages and Programming, 12th Colloquium, Nafplion, Greece, July 15-19, 1985, Proceedings*, volume 194 of *Lecture Notes in Computer Science*, pages 300–309. Springer, 1985. DOI (21)

[37] **Bakhadyr Khoussainov and Anil Nerode**. Automatic presentations of structures. *Logical and Computational Complexity. Selected Papers. Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, Indiana, USA, 13-16 October 1994*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer, 1994. DOI (20)

[38] **Jérémy Ledent**. Streaming string transducers (internship report), 2013. URL (5, 24, 26)

[39] **Arnaldo Mandel and Imre Simon**. On finite semigroups of matrices. *Theor. Comput. Sci.* 5(2):101–111, 1977. DOI (29, 30)

[40] **Anca Muscholl and Gabriele Puppis**. Equivalence of finite-valued streaming string transducers is decidable. *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, 122:1–122:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI (4, 5, 8–10)

[41] **Anca Muscholl and Gabriele Puppis**. The many facets of string transducers (invited talk). *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, 2:1–2:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI (3)

[42] **Brigitte Rozoy**. Outils et résultats pour les transducteurs boustrophedons. *RAIRO Theor. Informatics Appl.* 20(3):221–249, 1986. DOI (10)

[43] **Aleksi Saarela**. Systems of word equations, polynomials and linear algebra: A new approach. *Eur. J. Comb.* 47:1–14, 2015. DOI (11)

[44] **Jacques Sakarovitch**. A construction on finite automata that has remained hidden. *Theor. Comput. Sci.* 204(1-2):205–231, 1998. DOI (21)

[45] **Jacques Sakarovitch and Rodrigo de Souza**. Lexicographic decomposition of k-valued transducers. *Theory Comput. Syst.* 47(3):758–785, 2010. DOI (3)

[46] **Jacques Sakarovitch and Rodrigo de Souza**. On the decidability of bounded valuedness for transducers. *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 588–600. Springer, 2008. DOI (19)

[47] **Helmut Seidl**. Equivalence of finite-valued tree transducers is decidable. *Math. Syst. Theory*, 27(4):285–346, 1994. DOI (34)

[48] **John C. Shepherdson**. The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.* 3(2):198–200, 1959. DOI (5)

[49] **Richard Edwin Stearns and Harry B. Hunt III**. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.* 14(3):598–611, 1985. DOI (3)

[50] **Andreas Weber**. Decomposing a k-valued transducer into k unambiguous ones. *RAIRO Theor. Informatics Appl.* 30(5):379–413, 1996. DOI (3)

[51] **Andreas Weber**. Decomposing finite-valued transducers and deciding their equivalence. *SIAM J. Comput.* 22(1):175–202, 1993. DOI (3, 34)

[52] **Andreas Weber and Helmut Seidl**. On the degree of ambiguity of finite automata. *Theor. Comput. Sci.* 88(2):325–349, 1991. DOI (29, 30)

[53] **Di-De Yen and Hsu-Chun Yen**. On the decidability of the valuedness problem for two-way finite transducers. *Inf. Comput.* 285(Part):104870, 2022. DOI (6, 26, 33)