Sequentiality of String-to-Context Transducers

Pierre-Alain Reynier, Didier Villevalois

Sequentiality of String-to-Context Transducers

Pierre-Alain Reynier
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
pierre-alain.reynier@lis-lab.fr

Didier Villevalois
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
didier.villevalois@lis-lab.fr

Abstract

Transducers extend finite state automata with outputs, and describe transformations from strings to strings. Sequential transducers, which have a deterministic behaviour regarding their input, are of particular interest. However, unlike finite-state automata, not every transducer can be made sequential. The seminal work of Choffrut allows to characterise, amongst the functional one-way transducers, the ones that admit an equivalent sequential transducer.

In this work, we extend the results of Choffrut to the class of transducers that produce their output string by adding simultaneously, at each transition, a string on the left and a string on the right of the string produced so far. We call them the string-to-context transducers. We obtain a multiple characterisation of the functional string-to-context transducers admitting an equivalent sequential one, based on a Lipschitz property of the function realised by the transducer, and on a pattern (a new twinning property). Last, we prove that given a string-to-context transducer, determining whether there exists an equivalent sequential one is in coNP.

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

Keywords and phrases Transducers, Sequentiality, Twinning Property, Two-Way Transducers

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.123

Related Version An extended version can be found at `https://arxiv.org/abs/1902.11263`.

Funding This work has been funded by the DeLTA project (ANR-16-CE40-0007).

1 Introduction

Transducers are a fundamental model to describe programs manipulating strings. They date back to the very first works in theoretical computer science, and are already present in the pioneering works on finite state automata [25, 1]. While finite state automata are very robust w.r.t. modifications of the model such as non-determinism and two-wayness, this is not the case for transducers. These two extensions do affect the expressive power of the model. Non-determinism is a feature very useful for modelisation and specification purposes. However, when one turns to implementation, deriving a sequential, *i.e.* input-deterministic, transducer is a major issue. A natural and fundamental problem thus consists, given a (non-deterministic) transducer, in deciding whether there exists an equivalent sequential transducer. This problem is called the *sequentiality problem*.

In [12], Choffrut addressed this problem for the class of functional (one-way) finite state transducers, which corresponds to so-called *rational functions*. He proved a multiple characterisation of the transducers admitting an equivalent sequential transducer. This characterisation includes a machine-independent property, namely a Lipschitz property of the function realised by the transducer. It also involves a pattern property, namely the twinning property, that allows to prove that the sequentiality problem is decidable in polynomial time for the class of functional finite state transducers [27]. This seminal work has led to


developments on the sequentiality of finite state transducers [9, 8]. These results have also been extended to weighted automata [11, 22, 17] and to tree transducers [26]. See also [23] for a survey on sequentiality problems.

While the model of one-way transducers is now rather well-understood, a current challenge is to address the so-called class of *regular functions*, which corresponds to functions realised by two-way transducers. This class has attracted a lot of interest during the last years. It is closed under composition [13] and enjoys alternative presentations using logic [16], a deterministic one-way model equipped with registers, named streaming string transducers [2] (SST for short), a set of regular combinators akin to regular expressions for regular languages [5, 7, 14] as well as a class of functions operating on lists [10]. The class of regular functions is much more expressive than the class of rational functions, as it captures for instance the mirror image and the copy. Yet, it has good decidability properties: equivalence and type-checking are decidable in PSpace [21, 3], Similarly, functionality of non-deterministic SST is also decidable in PSpace [4]. We refer the interested reader to [19] for a recent survey. Intuitively, two-way finite state transducers (resp. SST) extend one-way finite state transducers with two important features: firstly, they can go through the input word both ways (resp. they can prepend and append words to registers), and secondly, they can perform multiple passes (resp. they can perform register concatenation).

In this paper, we lift the results of Choffrut [12] to a class of transducers that can perform the first of the two features mentioned above, thus falling strictly between the classes of rational and regular functions. More precisely, we consider non-deterministic transducers which, at each transition, extend the output word produced so far by prepending and appending two words to it. This operation can be defined as the extension of a word with a context, and we call these transducers the *string-to-context transducers*. However, it is important to notice that they still describe functions from strings to strings.

We characterise the functional string-to-context transducers that admit an equivalent *sequential* string-to-context transducer through i) a machine independent property: the function realised by the transducer satisfies a Lipschitz property that involves an original *factor distance* and ii) a pattern property of the transducer which we call *contextual twinning property*, and that generalises the twinning property to contexts. We also prove that the sequentiality problem for these transducers is in the class coNP.

A key technical tool of the result of [12] was a combinatorial analysis of the loops, showing that the output words of synchronised loops have conjugate primitive roots. For string-to-context transducers, the situation is more complex, as the combinatorics may involve the words of the two sides of the context. Intuitively, when these words do commute with the output word produced so far, it is possible for instance to move to the right a part of the word produced on the left. In order to prove our results, we thus dig into the combinatorics of contexts associated with loops, identifying different possible situations, and we then use this analysis to describe an original determinisation construction.

Our results also have a strong connection with the register minimisation problem for SST. This problem consists in determining, given an SST and a natural number k, whether there exists an equivalent SST with k registers. It has been proven in [15] that the problem is decidable for SST that can only append (and not prepend) words to registers, and the proof crucially relies on the fact that the $k = 1$ case exactly corresponds to the sequentiality problem of one-way finite state transducers. Hence, our results constitute a first step towards register minimisation for SST without register concatenation. The register minimisation problem for *non-deterministic* SST has also been studied in [6] for the case of concatenation-free SST. The targeted model being non-deterministic, the two problems are independent.

2 Models

Words, contexts and partial functions

Let A be a finite alphabet. The set of finite words (or strings) over A is denoted by A^*. The empty word is denoted by ϵ. The length of a word u is denoted by $|u|$. We say that a word u is a *prefix* (resp. *suffix*) of a word v if there exists a word y such that $uy = v$ (resp. $yu = v$). We say that two words $u, v \in A^*$ are *conjugates* if there exist two words $t_1, t_2 \in A^*$ such that $u = t_1 t_2$ and $v = t_2 t_1$. If this holds, we write $u \sim v$. The *primitive root* of a word $u \in A^*$, denoted $\rho(u)$, is the shortest word x such that $u = x^p$ for some $p \geqslant 1$. A word is said to be *primitive*, if it is equal to its primitive root. Given a word $u \in B^*$, we say that v is a *factor* of u if there exist words x, y such that $u = xvy$. For $n, m \in \mathbb{N}_{>0}$, we note by $\gcd(n, m)$ the greatest common divisor of n and m.

▶ **Lemma 1** (Fine and Wilf, [20], Chapter 9 of [24]). *Let $x, y \in A^*$ and $m, n \in \mathbb{N}$. If x^m and y^n have a common factor of length at least $|x| + |y| - \gcd(|x|, |y|)$, then their primitive roots are conjugates.*

Given two words $u, v \in A^*$, the *longest common prefix* (resp. *suffix*) of u and v is denoted by $\mathsf{lcp}(u, v)$ (resp. $\mathsf{lcs}(u, v)$). We define the *prefix distance* between u and v, denoted by $dist_p(u, v)$, as $|u| + |v| - 2|\mathsf{lcp}(u, v)|$.

Given two words $u, v \in B^*$, a *longest common factor* of u and v is a word w of maximal length that is a factor of both u and v. Note that this word is not necessarily unique. We denote such a word by $\mathsf{lcf}(u, v)$. The *factor distance* between u and v, denoted by $dist_f(u, v)$, is defined as $dist_f(u, v) = |u| + |v| - 2|\mathsf{lcf}(u, v)|$. This definition is correct as $|\mathsf{lcf}(u, v)|$ is independent of the choice of the common factor of maximal length.

Using a careful case analysis, we can prove that $dist_f$ is indeed a distance, the only difficulty lying in the subadditivity:

▶ **Lemma 2.** *$dist_f$ is a distance.*

Given a finite alphabet B, a *context* on B is a pair of words $(u, v) \in B^* \times B^*$. The set of contexts on B is denoted $\mathcal{C}(B)$. The empty context is denoted by c_ε. For a context $c = (u, v)$, we denote by $\overleftarrow{c}$ (resp. $\overrightarrow{c}$) its left (resp. right) component: $\overleftarrow{c} = u$ (resp. $\overrightarrow{c} = v$). The *length* of a context c is defined by $|c| = |\overleftarrow{c}| + |\overrightarrow{c}|$. The *lateralized length* of a context c is defined by $\|c\| = (|\overleftarrow{c}|, |\overrightarrow{c}|)$. For a context $c \in \mathcal{C}(B)$ and a word $w \in B^*$, we write $c[w]$ for the word $\overleftarrow{c}\, w\, \overrightarrow{c}$. We define the concatenation of two contexts $c_1, c_2 \in \mathcal{C}(B)$ as the context $c_1 c_2 = (\overleftarrow{c_1}\overleftarrow{c_2}, \overrightarrow{c_2}\overrightarrow{c_1})$. Last, given a context c and a word u, we denote by $c^{-1}[u]$ the unique word v such that $c[v] = u$, when such a word exists.

Given a set of contexts $C \subseteq \mathcal{C}(B)$, we denote by $\mathsf{lcc}(C)$ the longest common context of elements in C, defined as $\mathsf{lcc}(C) = (\mathsf{lcs}(\{\overleftarrow{c} \mid c \in C\}), \mathsf{lcp}(\{\overrightarrow{c} \mid c \in C\}))$. We also write $C.\mathsf{lcc}(C)^{-1} = \{c' \mid c'.\mathsf{lcc}(C) \in C\}$.

We consider two sets X, Y. Given $\Delta \subseteq X \times Y$, we let $\mathsf{dom}(\Delta) = \{x \in X \mid \exists y, (x, y) \in \Delta\}$. We denote the set of partial functions from X to Y as $\mathcal{F}(X, Y)$. Given $f \in \mathcal{F}(X, Y)$, we write $f : X \hookrightarrow Y$, and we denote by $\mathsf{dom}(f)$ its domain. When more convenient, we may also see elements of $\mathcal{F}(X, Y)$ as subsets of $X \times Y$. Last, given $\Delta \subseteq X \times Y$, we let $\mathsf{choose}(\Delta)$ denote some $\Delta' \in \mathcal{F}(X, Y)$ such that $\Delta' \subseteq \Delta$ and $\mathsf{dom}(\Delta) = \mathsf{dom}(\Delta')$.

String-to-Context and String-to-String Transducers

▶ **Definition 3.** *Let A, B be two finite alphabets. A string-to-context transducer (S2C for short) $\mathcal{T}$ from A^* to B^* is a tuple $(Q, t_{init}, t_{final}, T)$ where Q is a finite set of states,*

$t_{init} : Q \hookrightarrow \mathcal{C}(B)$ (resp. $t_{final} : Q \hookrightarrow \mathcal{C}(B)$) is the finite initial (resp. final) function, $T \subseteq Q \times A \times \mathcal{C}(B) \times Q$ is the finite set of transitions.

A state q is said to be *initial* (*resp. final*) if $q \in \mathsf{dom}(t_{init})$ (resp. $q \in \mathsf{dom}(t_{final})$). We depict as $\xrightarrow{c} q$ (resp. $q \xrightarrow{c}$) the fact that $t_{init}(q) = c$ (resp. $t_{final}(q) = c$). A run ρ from a state q_1 to a state q_k on a word $w = w_1 \cdots w_k \in A^*$ where for all i, $w_i \in A$, is a sequence of transitions: $(q_1, w_1, c_1, q_2), (q_2, w_2, c_2, q_3), \ldots, (q_k, w_k, c_k, q_{k+1})$. The *output* of such a run is the context $c = c_k \ldots c_2 c_1 \in \mathcal{C}(B)$, and is denoted by $\mathsf{out}(\rho)$. We depict this situation as $q_1 \xrightarrow{w|c} q_{k+1}$. The set of runs of $\mathcal{T}$ is denoted $\mathcal{R}(\mathcal{T})$. The run ρ is said to be *accepting* if q_1 is initial and q_{k+1} final. This string-to-context transducer $\mathcal{T}$ computes a relation $[\![\mathcal{T}]\!] \subseteq A^* \times B^*$ defined by the set of pairs $(w, edc[\varepsilon])$ such that there are $p, q \in Q$ with $\xrightarrow{c} p \xrightarrow{w|d} q \xrightarrow{e}$. Thus, even if its definition involves contexts on B, the semantics of $\mathcal{T}$ is a relation between words on A and words on B. Given an S2C $\mathcal{T} = (Q, t_{init}, t_{final}, T)$, we define the constant $M_{\mathcal{T}}$ as $M_{\mathcal{T}} = \max\{|c| \mid (p, a, c, q) \in T \text{ or } (q, c) \in t_{init} \cup t_{final}\}$. Given $\Delta : Q \hookrightarrow \mathcal{C}(B)$, we denote by $\mathcal{T}_\Delta$ the S2C obtained by replacing t_{init} with Δ. An S2C is *trimmed* if each of its states appears in some accepting run. W.l.o.g., we assume that the string-to-context transducers we consider are trimmed. An S2C $\mathcal{T}$ from A^* to B^* is *functional* if the relation $[\![\mathcal{T}]\!]$ is a function from A^* to B^*. An S2C $\mathcal{T} = (Q, t_{init}, t_{final}, T)$ is *sequential* if $\mathsf{dom}(t_{init})$ is a singleton and if for every transitions $(p, a, c, q), (p, a, c', q') \in T$, we have $q = q'$ and $c = c'$.

The classical model of finite-state transducers is recovered in the following definition:

▶ **Definition 4.** *Let A, B be two finite alphabets. A string-to-context transducer $\mathcal{T} = (Q, t_{init}, t_{final}, T)$ is a string-to-string transducer (S2S for short) from A^* to B^* if, for all $(q, c) \in t_{init} \cup t_{final}$, $\overleftarrow{c} = \varepsilon$, and for all $(q, a, c, q') \in T$, $\overleftarrow{c} = \varepsilon$.*

Notations defined for S2C hold for classical transducers as is. For an S2S, we write $\xrightarrow{w} q$ (resp. $q \xrightarrow{w}$, and $q \xrightarrow{u|w} q'$) instead of $\xrightarrow{(\varepsilon,w)} q$ (resp. $q \xrightarrow{(\varepsilon,w)}$, and $q \xrightarrow{u|(\varepsilon,w)} q'$).

Given an S2C $\mathcal{T} = (Q, t_{init}, t_{final}, T)$, we define its *right S2S*, denoted $\overrightarrow{\mathcal{T}}$, as the tuple $(Q, \overrightarrow{t_{init}}, \overrightarrow{t_{final}}, \overrightarrow{T})$ where, for all $q \in Q$, $\overrightarrow{t_{init}}(q) = \overrightarrow{t_{init}(q)}$ and $\overrightarrow{t_{final}}(q) = \overrightarrow{t_{final}(q)}$, and, for all $(p, a, c, q) \in T$, $(p, a, \overrightarrow{c}, q) \in \overrightarrow{T}$. Its *left S2S* $\overleftarrow{\mathcal{T}}$ is defined similarly, and by applying the mirror image on its output labels.

▶ **Example 5.** Two examples of S2C (not realisable by S2S) are depicted on Figure 1.

(a) $\mathcal{T}_{mirror}$ **(b)** $\mathcal{T}_{partition}$

Figure 1 1a Example of a S2C $\mathcal{T}_{mirror}$ computing the function $f_{mirror} : u_1 \ldots u_n \in \{a, b\}^* \mapsto u_n \ldots u_1$. 1b Example of a S2C $\mathcal{T}_{partition}$ computing the function $f_{partition} : u \in \{a, b\}^* \mapsto a^{|u|_a} b^{|u|_b}$.

3 Lipschitz and Twinning Properties

We recall the properties considered in [12], and the associated results.

▶ **Definition 6.** *We say that a function $f : A^* \hookrightarrow B^*$ satisfies the* Lipschitz property *if there exists $K \in \mathbb{N}$ such that $\forall u, v \in \mathsf{dom}(f), dist_p(f(u), f(v)) \leqslant K\, dist_p(u, v)$.*

▶ **Definition 7.** *We consider an S2S and $L \in \mathbb{N}$. Two states q_1 and q_2 are said to be* L-twinned *if for any two runs $\xrightarrow{w_1} p_1 \xrightarrow{u|x_1} q_1 \xrightarrow{v|y_1} q_1$ and $\xrightarrow{w_2} p_2 \xrightarrow{u|x_2} q_2 \xrightarrow{v|y_2} q_2$, where p_1 and p_2 are initial, we have for all $j \geqslant 0$, $dist_p(w_1 x_1 y_1^j, w_2 x_2 y_2^j) \leqslant L$. An S2S satisfies the* twinning property (TP) *if there exists $L \in \mathbb{N}$ such that any two of its states are L-twinned.*

▶ **Theorem 8** ([12])**.** *Let $\mathcal{T}$ be a functional S2S. The following assertions are equivalent:*
1. *there exists an equivalent sequential S2S,*
2. *$[\![\mathcal{T}]\!]$ satisfies the Lipschitz property,*
3. *$\mathcal{T}$ satisfies the twinning property.*

We present the adaptation of these properties to string-to-context transducers.

▶ **Definition 9.** *We say that $f : A^* \hookrightarrow B^*$ satisfies the* contextual Lipschitz property (CLip) *if there exists $K \in \mathbb{N}$ such that $\forall u, v \in \mathsf{dom}(f), dist_f(f(u), f(v)) \leqslant K\, dist_p(u, v)$.*

▶ **Definition 10.** *We consider an S2C and $L \in \mathbb{N}$. Two states q_1 and q_2 are said to be* L-contextually twinned *if for any two runs $\xrightarrow{c_1} p_1 \xrightarrow{u|d_1} q_1 \xrightarrow{v|e_1} q_1$ and $\xrightarrow{c_2} p_2 \xrightarrow{u|d_2} q_2 \xrightarrow{v|e_2} q_2$, where p_1 and p_2 are initial, we have for all $j \geqslant 0$, $dist_f(e_1^j d_1 c_1[\varepsilon], e_2^j d_2 c_2[\varepsilon]) \leqslant L$. An S2C satisfies the* contextual twinning property (CTP) *if there exists $L \in \mathbb{N}$ such that any two of its states are L-contextually twinned.*

4 Main Result

The main result of the paper is the following theorem, which extends to string-to-context transducers the characterisation of sequential transducers amongst functional ones.

▶ **Theorem 11.** *Let $\mathcal{T}$ be a functional S2C. The following assertions are equivalent:*
1. *there exists an equivalent sequential string-to-context transducer,*
2. *$[\![\mathcal{T}]\!]$ satisfies the contextual Lipschitz property,*
3. *$\mathcal{T}$ satisfies the contextual twinning property.*

Proof. The implications $1 \Rightarrow 2$ and $2 \Rightarrow 3$ are proved in Proposition 12 and Proposition 13 respectively. The implication $3 \Rightarrow 1$ is more involved, and is based on a careful analysis of word combinatorics of loops of string-to-context transducers satisfying the CTP. This analysis is summarised in Lemma 22 and used in Section 6 to describe the construction of an equivalent sequential S2C. ◀

▶ **Proposition 12.** *Let $\mathcal{T}$ be a sequential S2C realizing the function f. Then f satisfies the contextual Lipschitz property.*

Proof. We claim that f is context-Lipschitzian with coefficient $3M_{\mathcal{T}}$. Consider two input words u, v in the domain of f. If $u = v$, then the result is trivial. Otherwise, let $w = \mathsf{lcp}(u, v)$ and let $u = w.u'$, with $0 \leqslant |u'|$. Then we have $[\![\mathcal{T}]\!](u) = c_3 c_2 c_1[\epsilon]$ where c_1 is the context produced along w, c_2 the one produced along u', and c_3 is the final output context. Similarly, we can write (with $v = w.v'$, and $0 \leqslant |v'|$) $[\![\mathcal{T}]\!](v) = d_3 d_2 d_1[\epsilon]$. As $\mathcal{T}$ is sequential, we have $d_1 = c_1$. We also have $|c_3| \leqslant M_{\mathcal{T}}$, $|d_3| \leqslant M_{\mathcal{T}}$, $|c_2| \leqslant M_{\mathcal{T}}\ |u'|$ and $|d_2| \leqslant M_{\mathcal{T}}\ |v'|$. Finally, as $u \neq v$, we have $dist_p(u, v) = |u'| + |v'| \geqslant 1$ and we obtain:

$$dist_f(f(u), f(v)) \leqslant |c_3 c_2| + |d_3 d_2| \leqslant M_{\mathcal{T}}\ (2 + |u'| + |v'|) \leqslant 3M_{\mathcal{T}}\ dist_p(u, v)$$ ◀

▶ **Proposition 13.** *Let $\mathcal{T}$ be a functional S2C realizing the function f. If f satisfies the contextual Lipschitz property, then $\mathcal{T}$ satisfies the contextual twinning property.*

Proof. We consider an instance of the CTP and stick to the notations of Definition 10. We denote by n the number of states of $\mathcal{T}$. As $\mathcal{T}$ is trimmed, there exist runs $q_i \xrightarrow{w_i|f_i} r_i \xrightarrow{g_i}$, with $|w_i| \leqslant n$, for $i \in \{1,2\}$. We consider the input words $\alpha_j = uv^j w_1$ and $\beta_j = uv^j w_2$, for all $j \geqslant 0$. We have, for every j, $dist_p(\alpha_j, \beta_j) \leqslant |w_1| + |w_2| \leqslant 2n$.

The following property of $dist_f$ can be proven using a case analysis:

Fact. For every $w, w' \in B^*$, $c, c' \in \mathcal{C}(B)$, we have $dist_f(w, w') \leqslant dist_f(c[w], c'[w']) + |c| + |c'|$.

As f is K context-Lipschitzian, for some fixed K, we obtain, for all j:

$$\begin{aligned} dist_f(e_1^j d_1 c_1[\varepsilon], e_2^j d_2 c_2[\varepsilon]) &\leqslant dist_f(g_1 f_1 e_1^j d_1 c_1[\varepsilon], g_2 f_2 e_2^j d_2 c_2[\varepsilon]) + 2(n+1)M_{\mathcal{T}} \\ &\leqslant dist_f(f(\alpha_j), f(\beta_j)) + 2(n+1)M_{\mathcal{T}} \\ &\leqslant K\, dist_p(\alpha_j, \beta_j) + 2(n+1)M_{\mathcal{T}} \leqslant 2Kn + 2(n+1)M_{\mathcal{T}} \end{aligned}$$ ◀

5 Analysis of Loop Combinatorics

The classical twinning property forces the outputs of two runs reading the same input to only diverge by a finite amount. This constraint in turn makes for strong combinatorial bindings between runs involving loops: for two runs $\xrightarrow{w_1} p_1 \xrightarrow{u|x_1} q_1 \xrightarrow{v|y_1} q_1$ and $\xrightarrow{w_2} p_2 \xrightarrow{u|x_2} q_2 \xrightarrow{v|y_2} q_2$, we have $|y_1| = |y_2|$, and $\rho(y_1) \sim \rho(y_2)$. Similar behaviours are expected with string-to-context transducers and lead us to study the combinatorial properties of synchronised runs involving loops in those machines. Throughout this section, we consider a string-to-context transducer $\mathcal{T} = (Q, t_{init}, t_{final}, T)$ that satisfies the contextual twinning property.

5.1 Behaviours of Loops

We start with two examples illustrating how output contexts of synchronised loops can be modified to obtain an equivalent sequential S2C.

▶ **Example 14.** Figure 2a shows an example of a non-sequential functional S2C transducer $\mathcal{T}_1$. The contexts produced on loops around states q_1 and q_2 both commute with word a. This observation can be used to build an equivalent sequential S2C $\mathcal{D}_1$, depicted on Figure 2c. Figure 2b shows an example of a non-sequential functional S2C transducer $\mathcal{T}_2$ where output contexts are non-commuting, but can be slightly shifted so as to be aligned. This observation can be used to build an equivalent sequential S2C $\mathcal{D}_2$, depicted on Figure 2d.

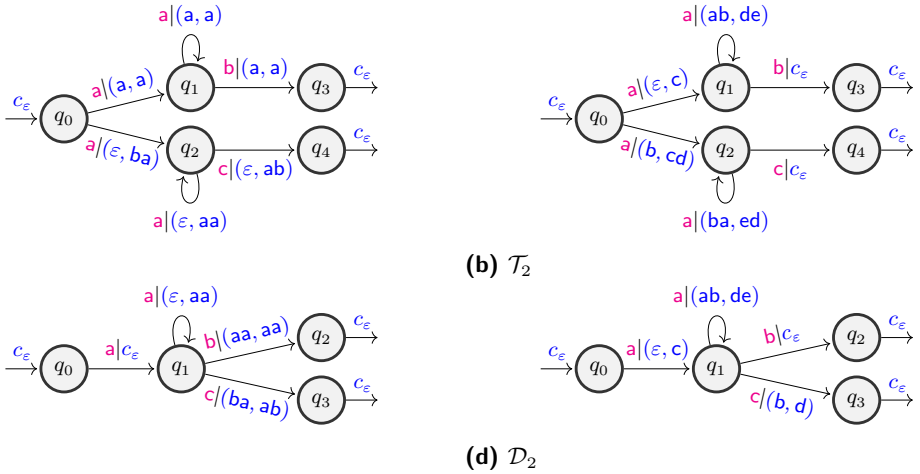
The following definition follows from the intuition drawn by the previous example.

▶ **Definition 15** (Lasso, Aligned/Commuting/Non-commuting lasso). *A lasso around a state q is a run ρ of the form $\xrightarrow{c} p \xrightarrow{u|d} q \xrightarrow{v|e} q$ with p an initial state. ρ is said to be productive, if $|e| \neq 0$. We say that ρ is:*

- *aligned w.r.t. f and w, for some $f \in \mathcal{C}(B)$ and $w \in B^*$, denoted as $(f,w)-$aligned, if there exists a context $g \in \mathcal{C}(B)$ such that for all $i \in \mathbb{N}$, $e^i dc[\varepsilon] = gf^i[w]$.*
- *commuting w.r.t. x, for some $x \in B^+$, denoted as $x-$commuting, if there exists a context $f \in \mathcal{C}(B)$ such that for all $i \in \mathbb{N}_{>0}$, there exists $k \in \mathbb{N}$ such that $e^i dc[\varepsilon] = f[x^k]$.*
- *non-commuting if there exists no word $x \in B^+$ such that ρ is commuting w.r.t x.*

Two lassos $\xrightarrow{c_1} p_1 \xrightarrow{u_1|d_1} q_1 \xrightarrow{v_1|e_1} q_1$ and $\xrightarrow{c_2} p_2 \xrightarrow{u_2|d_2} q_2 \xrightarrow{v_2|e_2} q_2$ are said to be synchronised if $u_1 = u_2$ and $v_1 = v_2$. They are said to be strongly balanced if $\|e_1\| = \|e_2\|$.

Given an integer $k \geqslant 1$, we consider the k-th power of $\mathcal{T}$, that we denote by $\mathcal{T}^k$. A run in $\mathcal{T}^k$ naturally corresponds to k synchronised runs in $\mathcal{T}$, *i.e.* on the same input word. We lift the notion of lasso to $\mathcal{T}^k$, and we denote them by $H_1 H_2$, where H_1 starts in initial states and



(a) $\mathcal{T}_1$ **(b)** $\mathcal{T}_2$

(c) $\mathcal{D}_1$ **(d)** $\mathcal{D}_2$

Figure 2 2a An S2C $\mathcal{T}_1$ computing the function that maps $a^n b$ to a^{2n+2} and $a^n c$ to $ba^{2n}b$. 2c A sequential S2C $\mathcal{D}_1$ equivalent to $\mathcal{T}_1$. 2b An S2C $\mathcal{T}_2$ computing the function that maps $a^n b$ to $(ab)^{n-1}c(de)^{n-1}$ and $a^n c$ to $b(ab)^{n-1}c(de)^{n-1}d$. 2d A sequential S2C $\mathcal{D}_2$ equivalent to $\mathcal{T}_2$.

ends in some state $q = (q_i)_{i\in\{1,\dots,k\}} \in Q^k$, and H_2 is a loop around state q. In the sequel, we will only consider lassos such that q contains pairwise distinct states ($q_i \neq q_j$ for all $i \neq j$). Those lassos are included in the lassos in $\mathcal{T}^{\leqslant|Q|} = \cup_{1\leqslant k\leqslant|Q|}\mathcal{T}^k$.

The intuition given by Example 14 is formalised in the following Lemma:

▶ **Lemma 16.** *Let $H_1H_2 = (\rho_j)_{j\in\{1,\dots,k\}}$ a lasso in $\mathcal{T}^k$, for some $1 \leqslant k \leqslant |Q|$. We write $\rho_j : \xrightarrow{c_j} p_j \xrightarrow{u_1|d_j} q_j \xrightarrow{u_2|e_j} q_j$ for each j. Then there exists an integer $m \in \mathbb{N}$ such that $|e_j| = m$ for all $j \in \{1,\dots,k\}$. If $m > 0$, we say that the lasso H_1H_2 is productive, and:*

- *either there exists $x \in B^+$ primitive such that ρ_j is $x-$commuting for all $j \in \{1,\dots,k\}$. In this case, we say that the lasso H_1H_2 is $x-$commuting, and we let $\mathsf{pow}_c(x, H_1, H_2) = m/|x|$ and $\mathsf{split}_c(x, H_1, H_2) = \{(q_j, f_j) \mid j \in \{1,\dots,k\}\}$ where $f_j \in \mathcal{C}(B)$ is such that $\forall\alpha \in \mathbb{N}, e_j^\alpha d_j c_j[\varepsilon] = f_j[x^{\alpha\ \mathsf{pow}_c(x,H_1,H_2)}]$.*[1]
- *or there exist $f \in \mathcal{C}(B)$ and $w \in B^*$ such that ρ_j is non-commuting and $(f, w)-$aligned for all $j \in \{1,\dots,k\}$. In this case, we say that the lasso H_1H_2 is $(f, w)-$aligned, and we let $\mathsf{split}_{nc}(f, w, H_1, H_2) = \{(q_j, g_j) \mid j \in \{1,\dots,k\}\}$ where $g_j \in \mathcal{C}(B)$ is such that $\forall\alpha \in \mathbb{N}, e_j^\alpha d_j c_j[\varepsilon] = g_j f^\alpha[w]$.*[1]

Proof Sketch. As $\mathcal{T}$ satisfies the CTP, the outputs must grow at the same pace when the loops are pumped. This entails that the lengths of the e_j must be equal. Next, the result is proved by considering two productive synchronised lassos, with loops producing respectively e_1 and e_2. If they are not strongly balanced or one of them is $x-$commuting, for some $x \in B^+$, then, using the result of Fine and Wilf (Lemma 1) between $\overleftarrow{e_1}, \overleftarrow{e_2}, \overrightarrow{e_1}$ and $\overrightarrow{e_2}$, we can prove that the other one is also $x-$commuting. Otherwise, they are both non-commuting and strongly balanced. Using again Lemma 1 but first between $\overleftarrow{e_1}$ and $\overleftarrow{e_2}$, and then between $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$, we prove that there exist $f \in \mathcal{C}(B)$ and $w \in B^*$ such that ρ_1 and ρ_2 are $(f, w)-$aligned. Finally, the result is lifted to k productive synchronised lassos. ◀

[1] Because we only consider lassos around pairwise distinct states, both $\mathsf{split}_c(x, H_1, H_2)$ and $\mathsf{split}_{nc}(f, w, H_1, H_2)$ are partial functions from Q to $\mathcal{C}(B)$.

▶ **Example 17.** We consider the example S2C in Figure 2. The lasso in $\mathcal{T}_1^2$ around (q_1, q_2) is $\mathsf{a}-$commuting. We can compute a pow_c of 2 and $\{(q_1, (\mathsf{a}, \mathsf{a})), (q_2, (\mathsf{b}, \mathsf{a}))\}$ as a possible split_c. The lasso in $\mathcal{T}_2^2$ around (q_1, q_2) is $((\mathsf{ab}, \mathsf{de}), \mathsf{c})-$aligned. We can compute $\{(q_1, c_\varepsilon), (q_2, (\mathsf{b}, \mathsf{d}))\}$ as a possible split_{nc}.

5.2 Analysis of Loops Consecutive to a Productive Loop

Consider a run that contains two consecutive productive loops. We can observe that the type (commuting or non-commuting) of the lasso involving the first loop impacts the possible types of the lasso involving the second loop. For instance, it is intuitive that a non-commuting lasso cannot be followed by a commuting lasso. Similarly, an $x-$commuting lasso cannot be followed by a $y-$commuting lasso, if x and y are not conjugates. We will see that loops following a first productive loop indeed satisfy stronger combinatorial properties. The following definition characterises their properties.

▶ **Definition 18** (Strongly commuting/Strongly aligned lasso)**.** *Let ρ be a productive lasso $\xrightarrow{c} p \xrightarrow{u|d} q \xrightarrow{v|e} q$ and $x \in B^+$. We say that ρ is:*

- *strongly commuting w.r.t. x, denoted as* strongly$-x-$commuting*, if there exists a context $f \in \mathcal{C}(B)$ such that for all $i, j \in \mathbb{N}_{>0}$, there exists $k \in \mathbb{N}$ such that $e^i dc[x^j] = f[x^k]$.*
- *strongly aligned w.r.t. g, f and x, denoted as* strongly$-(g, f, x)-$aligned*, if there exists a context $h \in \mathcal{C}(B)$ such that for all $i, j \in \mathbb{N}$, $e^j dc[x^i] = hg^j f[x^i]$.*

The following Lemma states the properties of a lasso consecutive to a commuting lasso. To prove it, we proceed as for Lemma 16 by proving the result first for two runs and then lifting it to k runs. The case of two runs is obtained by distinguishing whether they are strongly balanced or not, and using Lemma 1.

▶ **Lemma 19.** *Let H_1H_2 a productive $x-$*commuting *lasso in $\mathcal{T}^{\leqslant |Q|}$, for some $x \in B^+$. Let $\Delta = \mathsf{split}_c(x, H_1, H_2)$ and $H_3H_4 = (\rho_j)_{j\in\{1,\dots,k\}}$ a productive lasso in $\mathcal{T}_\Delta^k$, for some $1 \leqslant k \leqslant |Q|$. We write $\rho_j : \xrightarrow{c_j} p_j \xrightarrow{u_1|d_j} q_j \xrightarrow{u_2|e_j} q_j$ for each j. Then:*

- *either every ρ_j is* strongly$-x-$commuting*: we say that H_3H_4 is* strongly$-x-$commuting*,*
- *or there exist $g, h \in \mathcal{C}(B)$ such that every ρ_j is* strongly$-(h, g, x)-$aligned*. In this case, we say that H_3H_4 is* strongly$-(h, g, x)-$aligned *and we let $\mathsf{extract}_{nc}(h, g, x, \Delta, H_3, H_4) = \{(q_j, h_j) \mid j \in \{1, \dots, k\}\}$ where $h_j \in \mathcal{C}(B)$ is s.t. $\forall \alpha, \beta \in \mathbb{N}, e_j^\alpha d_j c_j[x^\beta] = h_j h^\alpha g[x^\beta]$.*

The following Lemma states that once a non-commuting loop is encountered, then the alignment of production is fixed, *i.e.* no transfer between left and right productions is possible anymore. Hence, the left and right S2S derived from the S2C both satisfy the twinning property:

▶ **Lemma 20.** *Let H_1H_2 be a productive non-commuting lasso that is either*

- *$(f, w)-$*aligned *in $\mathcal{T}^{\leqslant |Q|}$, for some $f \in \mathcal{C}(B)$ and $w \in B^*$, and $\Delta' = \mathsf{split}_{nc}(f, w, H_1, H_2)$,*
- *or* strongly$-(g, f, x)-$aligned *in $\mathcal{T}_\Delta^{\leqslant |Q|}$, for some $g, f \in \mathcal{C}(B)$ and $\Delta \in \mathcal{F}(Q, \mathcal{C}(B))$, and $\Delta' = \mathsf{extract}_{nc}(g, f, x, \Delta, H_1, H_2)$.*

Then $\overleftarrow{\mathcal{T}_{\Delta'}}$ and $\overrightarrow{\mathcal{T}_{\Delta'}}$ both satisfy the twinning property.

5.3 A Two-loop Pattern Property

The following 2-loop property summarises the combinatorial properties of the synchronised runs involving loops in string-to-context transducers that satisfy the CTP.

▶ **Definition 21** (2-loop property). *Given four runs H_1, H_2, H_3, H_4 in $\mathcal{T}^{\leqslant |Q|}$, such that H_1H_2 and $(H_1H_3)H_4$ are lassos in $\mathcal{T}^{\leqslant |Q|}$, we say that they satisfy the 2-loop property if:*

1. *H_1H_2 is*
   - **a.** *either non productive,*
   - **b.** *or productive and $x-$*commuting*, for some $x \in B^+$,*
   - **c.** *or productive, non-commuting and $(f,w)-$*aligned*, for some $f \in \mathcal{C}(B)$ and $w \in B^*$.*
2. *if H_1H_2 is productive and $x-$*commuting*, we let $\Delta = \mathsf{split}_c(x, H_1, H_2)$, then H_3H_4 is a lasso in $\mathcal{T}_\Delta^{\leqslant |Q|}$. If productive then it is:*
   - **a.** *either* strongly$-x-$commuting*,*
   - **b.** *or non-commuting and* strongly$-(h,g,x)-$aligned*, for some $g, h \in \mathcal{C}(B)$. We let $\Delta' = \mathsf{extract}_{nc}(h, g, x, \Delta, H_3, H_4)$, then $\overleftarrow{\mathcal{T}_{\Delta'}}$ and $\overrightarrow{\mathcal{T}_{\Delta'}}$ both satisfy the twinning property.*
3. *if H_1H_2 is productive, non-commuting and $(f,w)-$*aligned*, we let $\Delta = \mathsf{split}_{nc}(f, w, H_1, H_2)$, then $\overleftarrow{\mathcal{T}_\Delta}$ and $\overrightarrow{\mathcal{T}_\Delta}$ both satisfy the twinning property.*

A string-to-context transducer $\mathcal{T}$ is said to satisfy the 2-loop property if for all runs H_1, H_2, H_3, H_4 as above, they satisfy the 2-loop property.

As a consequence of Lemmas 16, 19 and 20, we have:

▶ **Lemma 22.** *If an S2C $\mathcal{T}$ satisfies the CTP then it satisfies the 2-loop property.*

6 Determinisation

Throughout this section, we consider a string-to-context transducer $\mathcal{T} = (Q, t_{init}, t_{final}, T)$ from A^* to B^* that satisfies the 2-loop property. Intuitively, our construction stores the set of possible runs of $\mathcal{T}$, starting in an initial state, on the input word read so far. These runs are incrementally simplified by erasing synchronised loops, and by replacing a prefix by a partial function $\Delta : Q \hookrightarrow \mathcal{C}(B)$. These simplifications are based on the 2-loop property.

Observation It is worth noticing that, as $\mathcal{T}$ is functional, if two runs reach the same state, it is safe to keep only one of them. This allows us to maintain a set of at most $|Q|$ runs.

Notations Given $\Delta \in \mathcal{F}(Q, \mathcal{C}(B))$, $c \in \mathcal{C}(B)$, $w \in B^*$, $a \in A$ and $H \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})$, we define the following notations and operations:

- $\Delta c = \{(q, dc) \mid (q,d) \in \Delta\}$,
- $\Delta[w] = \{(q, d[w]) \mid (q,d) \in \Delta\}$,
- $\Delta \bullet a = \mathsf{choose}(\{(q', dc) \mid (q,c) \in \Delta \text{ and } q \xrightarrow{a|d} q'\})$,
- $H \bullet a \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})$ is the run obtained by extending runs of H with consecutive transitions of $\mathcal{T}$ associated with input symbol a, and by eliminating runs so as to ensure that runs reach pairwise distinct states of $\mathcal{T}$,
- $\Delta \bullet H = \mathsf{choose}(\{(q', dc) \mid (q,c) \in \Delta \text{ and there is a run } \rho : q \xrightarrow{x|d} q' \in H\})$,
- $id_\Delta = (q_i)_{1 \leqslant i \leqslant k} \in \mathcal{R}(\mathcal{T}^k)$, for some enumeration $\{q_1, \ldots, q_k\}$ of $\mathsf{dom}(\Delta)$.

Construction We define an equivalent deterministic string-to-context transducer $\overline{\mathcal{D}} = (\overline{Q}, \overline{t_{init}}, \overline{t_{final}}, \overline{T})$, and we denote by $\mathcal{D}$ its trim part. While $\overline{\mathcal{D}}$ may have infinitely many states, we will prove that $\mathcal{D}$ is finite. Formally, we define $\overline{Q} = \overline{Q}_{\mathsf{start}} \uplus \overline{Q}_{\mathsf{com}} \uplus \overline{Q}_{\neg\mathsf{com}}$ where:

- $\overline{Q}_{\mathsf{start}} = \{(\varepsilon, t_{init}, H) \mid H \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})\}$
- $\overline{Q}_{\mathsf{com}} = \{(x, \Delta, H) \mid x \in B^+, \Delta \in \mathcal{F}(Q, \mathcal{C}(B)), H \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})\}$
- $\overline{Q}_{\neg\mathsf{com}} = \{(\bot, \Delta, id_\Delta) \mid \Delta \in \mathcal{F}(Q, \mathcal{C}(B))\}$.

By definition, we have $\overline{Q} \subseteq (B^* \cup \{\bot\}) \times \mathcal{F}(Q, \mathcal{C}(B)) \times \mathcal{R}(\mathcal{T}^{\leqslant |Q|}) = \overline{Q}_\infty$. Given $\overline{q} = (x, \Delta, H) \in \overline{Q}_\infty$, we let $\Delta_{\overline{q}} = \Delta \bullet H \in \mathcal{F}(Q, \mathcal{C}(B))$. An invariant of our construction is that every starting state of a run in H belongs to $\mathsf{dom}(\Delta)$.

Intuitively, the semantics of a state $\overline{q} = (x, \Delta, H) \in \overline{Q}$ can be understood as follows: x is used to code the type of state ($\overline{Q}_{\mathsf{start}}$, $\overline{Q}_{\mathsf{com}}$ or $\overline{Q}_{\neg\mathsf{com}}$), and Δ and H are used to represent the runs that remain to be executed to faithfully simulate the runs of $\mathcal{T}$ on the input word u read so far. As we have seen in the previous section, loops may either be commuting, allowing to shift some parts of the output from one side of the context to the other side, or they are non-commuting, and then should be aligned, forbidding such modifications. Intuitively, states in $\overline{Q}_{\mathsf{start}}$ correspond to situations in which no productive loop has been encountered yet. States in $\overline{Q}_{\mathsf{com}}$ (with $x \in B^+$) correspond to situations in which only x-commuting loops have been encountered. States in $\overline{Q}_{\neg\mathsf{com}}$ correspond to situations in which a non-commuting loop has been encountered. A representation of $\mathcal{D}$ is given in Figure 3.

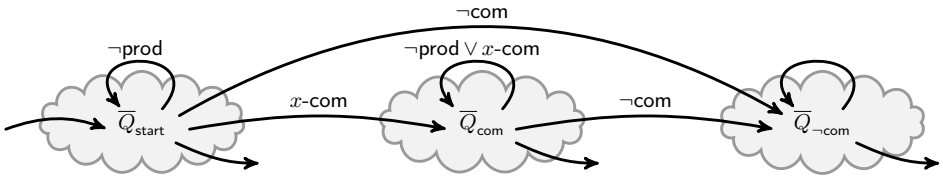


Figure 3 A schematic representation of states and transitions of $\mathcal{D}$.

Initial and final states They are defined as follows:

- $\overline{t_{init}} = \{(\overline{i}, c_\varepsilon)\}$ where $\overline{i} = (\varepsilon, t_{init}, id_{t_{init}}) \in \overline{Q}_{\mathsf{start}}$
- $\overline{t_{final}} = \mathsf{choose}(\{(\overline{q}, dc) \mid \overline{q} \in \overline{Q}, (p, c) \in \Delta_{\overline{q}}, (p, d) \in t_{final}\})$

Transitions They are defined as follows:

- $\overline{\mathcal{D}} = \{\overline{p} \xrightarrow{a|c} \overline{q} \mid \overline{p} = (x, \Delta, H) \in \overline{Q}, a \in A \text{ and } (\overline{q}, c) = \textsc{simplify}((x, \Delta, H \bullet a))\}$

Intuitively, a transition of $\overline{\mathcal{D}}$ leaving some state $\overline{p} = (x, \Delta, H) \in \overline{Q}$ with letter $a \in A$ aims at first extending H with a, obtaining the new set of runs $H \bullet a$, and then simplifying this set of runs by removing loops, using the function SIMPLIFY. This function is implemented as Algorithm 2, which calls Algorithm 1 to remove all loops of $H \bullet a$ one by one. Depending on the type of the loop encountered, the type of the state is updated.

We first define EXTEND_WITH_LOOP$(\overline{p}, H_2)$ in Algorithm 1 that takes as input a state $\overline{p} = (x, \Delta, H_1) \in \overline{Q}_{\mathsf{start}} \cup \overline{Q}_{\mathsf{com}}$ and a run H_2 in $\mathcal{T}^{\leqslant |Q|}$ such that $H_1 H_2$ is a lasso in $\mathcal{T}_\Delta^{\leqslant |Q|}$. The algorithm enumerates the possible cases for the type of this lasso, depending on the type of $\overline{p}$. This enumeration strongly relies on the 2-loop property. Depending on the case, the loop is processed, and a pair composed of a new state and a context is returned. This context will be part of the output associated with the transition. By a case analysis, we prove:

▶ **Lemma 23.** *Let $\overline{p} = (x, \Delta, H_1) \in \overline{Q}_{start} \cup \overline{Q}_{com}$ and $H_2 \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})$ such that $H_1 H_2$ is a lasso in $\mathcal{T}_\Delta^{\leqslant |Q|}$. We let $(\overline{q}, c) =$ EXTEND_WITH_LOOP$(\overline{p}, H_2)$.*

- *If $x = \varepsilon$ then $(\Delta_{\overline{p}} \bullet H_2)[\varepsilon] = \Delta_{\overline{q}} c[\varepsilon]$.*
- *If $x \in B^+$ then for all $k \in \mathbb{N}$, $(\Delta_{\overline{p}} \bullet H_2)[x^k] = \Delta_{\overline{q}} c[x^k]$.*

We then define SIMPLIFY$(\overline{p})$ in Algorithm 2 that takes as input a state $\overline{p} \in \overline{Q}_\infty$ (we need to consider $\overline{Q}_\infty$ as input and not only $\overline{Q}$ because of the recursive calls) and returns a pair composed of a new state and a context. Intuitively, it recursively processes the lassos present

Algorithm 1 Extending a state $\overline{p} = (x, \Delta, H_1) \in \overline{Q}_{\mathsf{start}} \cup \overline{Q}_{\mathsf{com}}$ with $H_2 \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})$ s.t. H_1H_2 is a lasso in $\mathcal{T}_\Delta^{\leqslant |Q|}$.

1: **function** EXTEND_WITH_LOOP($\overline{p}, H_2$)
2: **if** H_2 is non-productive **then**
3: **return** $(\overline{p}, c_\varepsilon)$
4: **else if** $\overline{p} = (\varepsilon, t_{init}, H_1)$ **then**
5: **if** H_1H_2 is $x-$commuting, for some $x \in B^+$, **then**
6: **let** $\Delta = \mathsf{split}_c(x, H_1, H_2)$ and $k = \mathsf{pow}_c(x, H_1, H_2)$
7: **return** $((x, \Delta, id_\Delta), (\varepsilon, x^k))$
8: **else if** H_1H_2 is $(f, w)-$aligned, for some $f \in \mathcal{C}(B)$ and $w \in B^*$, **then**
9: **let** $\Delta = \mathsf{split}_{nc}(f, w, H_1, H_2)$
10: **return** $((\bot, \Delta, id_\Delta), f \cdot (\varepsilon, w))$
11: **end if**
12: **else if** $\overline{p} = (x, \Delta_0, H_1)$, where $x \in B^+$, **then**
13: **if** H_1H_2 is strongly$-x-$commuting **then**
14: **let** $k = |\mathsf{out}(H_2)|/|x|$
15: **return** $(\overline{p}, (\varepsilon, x^k))$
16: **else if** H_1H_2 is strongly$-(g, f, x)-$aligned, for some $g, f \in \mathcal{C}(B)$, **then**
17: **let** $\Delta = \mathsf{extract}_{nc}(g, f, x, \Delta_0, H_1, H_2)$
18: **return** $((\bot, \Delta, id_\Delta), gf)$
19: **end if**
20: **end if**
21: **end function**

in the runs stored by the state $\overline{p}$, by using calls to the previous algorithm. The following result is proved by induction, using Lemma 23:

▶ **Lemma 24.** *Let $\overline{p} = (x, \Delta, H) \in \overline{Q}_\infty$ and $(\overline{q}, c) = \text{SIMPLIFY}(\overline{p})$. Then $\overline{q} \in \overline{Q}$ and we have:*

- *If $x = \varepsilon$ then $\Delta_{\overline{p}}[\varepsilon] = \Delta_{\overline{q}}c[\varepsilon]$.*
- *If $x \in B^+$ then for all $k \in \mathbb{N}$, $\Delta_{\overline{p}}[x^k] = \Delta_{\overline{q}}c[x^k]$.*
- *If $x = \bot$ then $\Delta_{\overline{p}} = \Delta_{\overline{q}}c$.*

▶ **Theorem 25.** *$\mathcal{D}$ is a finite sequential string-to-context transducer equivalent to $\mathcal{T}$.*

Proof Sketch. First observe that $\mathcal{D}$ is sequential. The correctness of $\mathcal{D}$ is a consequence of the following property, that we prove using Lemma 24 and an induction on $|u|$: for all $u \in A^*$, if we have $\overline{i} \xrightarrow{u|c} \overline{q}$ in $\mathcal{D}$, then $\Delta_{\overline{q}}c[\varepsilon] = (t_{init} \bullet u)[\varepsilon]$. Last, we prove that $\mathcal{D}$ is finite. By construction, for every state $\overline{q} = (x, \Delta, H)$ of $\mathcal{D}$, H contains no loop, hence its length is bounded by $|Q|^{|Q|}$. This can be used to bound the size of x, as well as the size of Δ, for states in $\overline{Q}_{\mathsf{start}} \cup \overline{Q}_{\mathsf{com}}$. The case of states in $\overline{Q}_{\neg\mathsf{com}}$ is different: when such a state $(\bot, \Delta, id_\Delta)$ is reached, then by the 2-loop property, the transducers $\overleftarrow{\mathcal{T}_\Delta}$ and $\overrightarrow{\mathcal{T}_\Delta}$ both satisfy the (classical) twinning property. It remains to observe that the operations performed on Line 24 precisely correspond to two determinisations of [12], on both sides of the S2C. ◀

7 Decision

In this section, we prove the following result:

Algorithm 2 Simplifying a state $\overline{p} = (x, \Delta, H) \in \overline{Q}_\infty$.

22: **function** SIMPLIFY($\overline{p}$)
23: **if** $\overline{p} = (\bot, \Delta, H)$ **then**
24: **let** $\Delta' = \Delta \bullet H$, $c = \mathsf{lcc}(\Delta')$ and $\overline{q} = (\bot, \Delta'.c^{-1}, id_{\Delta'})$
25: **return** $(\overline{q}, c)$
26: **else if** $\overline{p} = (x, \Delta, H_1H_2H_3)$, where $x \in B^*$ and H_2 is the first loop in H, **then**
27: **let** $\overline{q} = (x, \Delta, H_1)$
28: **let** $(\overline{r}, c) =$ EXTEND_WITH_LOOP$(\overline{q}, H_2)$ with $\overline{r} = (x', \Delta', H')$
29: **let** $(\overline{s}, d) =$ SIMPLIFY$((x', \Delta', H'.H_3))$
30: **return** $(\overline{s}, dc)$
31: **else**
32: **return** $(\overline{p}, c_\varepsilon)$
33: **end if**
34: **end function**

▶ **Theorem 26.** *Given a string-to-context transducer, determining whether there exists an equivalent sequential string-to-context transducer is in coNP.*

In order to show this result, we introduce a restriction of the 2-loop property:

▶ **Definition 27** (small-2-loop property)**.** *A string-to-context transducer $\mathcal{T}$ is said to satisfy the small-2-loop property if, for all runs $H_1, H_2, H_3, H_4 \in \mathcal{T}^2$ with $|H_i| \leqslant |Q|^2$ for each i, H_1H_2, $H_1H_3H_4$ are lassos and they satisfy the 2-loop property (in the sense of Definition 21).*

By definition, if a string-to-context transducer satisfies the 2-loop property then it also satisfies the small-2-loop property. We will show that the two properties are equivalent.

▶ **Lemma 28.** *If a string-to-context transducer $\mathcal{T}$ satisfies the small-2-loop property then $[\![\mathcal{T}]\!]$ satisfies the contextual Lipschitz property.*

Proof Sketch. We claim there exists $K \in \mathbb{N}$ such that for every pair of synchronised runs $H : \xrightarrow{(c_0,d_0)} (p_0, q_0) \xrightarrow{u|(c_1,d_1)} (p_1, q_1)$ in $\mathcal{T}^2$, we have $dist_f(c_1c_0[\varepsilon], d_1d_0[\varepsilon]) \leqslant K$. The result then easily follows. To prove this claim, we apply the main procedure SIMPLIFY (see Section 6) to the state $\overline{p} = (\varepsilon, t_{init}, H)$. This procedure can indeed be applied: as it always processes the *first* loop (see Line 29), the lassos considered satisfy the premises of the small-2-loop property. The claim follows from the proof of finiteness of $\mathcal{D}$. ◀

Proof Sketch of Theorem 26. By Theorem 11 and Lemma 28, $\mathcal{T}$ admits an equivalent sequential S2C transducer iff $\mathcal{T}$ satisfies the small-2-loop property (see also Figure 4). Thus, we describe a procedure to decide whether $\mathcal{T}$ satisfies the small-2-loop property.

The procedure first non-deterministically guesses a counter-example to the small-2-loop property and then verifies that it is indeed a counter-example. By definition of the small-2-loop property, the counter-example can have finitely many shapes. Those shapes require the verification of the properties of the involved lassos: being productive or not, being commuting or not, being aligned or not, satisfying the (classical) twinning property, etc.

Verifying that a lasso in $\mathcal{T}^2$ is not commuting (resp. not aligned) boils down to checking whether there exists no $x \in B^+$ such that the lasso is $x-$commuting (resp. no $f \in \mathcal{C}(B)$ and $w \in B^*$ such that the lasso is $(f, w)-$aligned). In both cases, the search space for the words x, w and context f can be narrowed down to factors of the output contexts of the given lasso. Thus these verifications can be done in polynomial time. The classical twinning property can

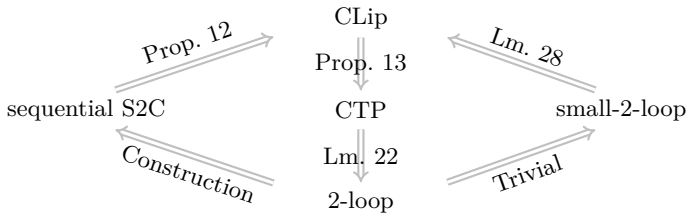


Figure 4 Overview of the equivalent properties we consider.

also be checked in polynomial time. As a summary, we can show that the verifications for all the shapes can be done in polynomial time. Furthermore, all the shapes are of polynomial size, by definition of the small-2-loop property, yielding the result. ◀

Note that if one can express in the logic of [18] that a lasso in $\mathcal{T}^2$ is not commuting (resp. not aligned), then this would show that the problem can be solved in polynomial time. However, this seems difficult because of the universal quantification on the factor x.

8 Conclusion

We have proposed a multiple characterisation of string-to-context transducers that admit an equivalent sequential S2C, including a machine independent property, a pattern property, as well as a "small" pattern property allowing to derive a decision procedure running in non-deterministic polynomial time. All these equivalences are summarised in Figure 4. Future work includes a lower bound for the complexity of the problem, the extension of this work to the register minimisation problem for streaming string transducers without register concatenation, and the extension of our results to infinite words.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. A general theory of translation. *Mathematical Systems Theory*, 3(3):193–221, 1969.

2 Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

3 Rajeev Alur and Pavol Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proc. of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*, pages 599–610. ACM, 2011.

4 Rajeev Alur and Jyotirmoy V. Deshmukh. Nondeterministic streaming string transducers. In *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.

5 Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In *CSL-LICS '14*, pages 9:1–9:10. ACM, 2014.

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

7 Nicolas Baudru and Pierre-Alain Reynier. From two-way transducers to regular function expressions. In *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 96–108. Springer, 2018.

8 Marie-Pierre Béal and Olivier Carton. Determinization of transducers over finite and infinite words. *Theoretical Computer Science*, 289(1):225–251, 2002.

9 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1):45–63, 2003.

10 Mikolaj Boja'nczyk, Laure Daviaud, and Shankara Narayanan Krishna. Regular and first-order list functions. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 125–134. ACM, 2018.

11 Adam L. Buchsbaum, Raffaele Giancarlo, and Jeffery Westbrook. On the determinization of weighted finite automata. *SIAM J. Comput.*, 30(5):1502–1531, 2000.

12 Christian Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977.

13 Michal Chytil and Vojtech Jákl. Serial composition of 2-way finite-state transducers and simple programs on strings. In *Automata, Languages and Programming, Fourth Colloquium, University of Turku, Finland, July 18-22, 1977, Proceedings*, volume 52 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 1977.

14 Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. Regular transducer expressions for regular transformations. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 315–324. ACM, 2018.

15 Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. A generalised twinning property for minimisation of cost register automata. In *LICS '16*, pages 857–866. ACM, 2016.

16 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.

17 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Quantitative languages defined by functional automata. *Logical Methods in Computer Science*, 11(3), 2015.

18 Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. A pattern logic for automata with outputs. In *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 304–317. Springer, 2018.

19 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *SIGLOG News*, 3(3):4–19, 2016.

20 N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16:109–114, 1965.

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

22 Daniel Kirsten and Ina Mäurer. On the determinization of weighted automata. *Journal of Automata, Languages and Combinatorics*, 10(2/3):287–312, 2005.

23 Sylvain Lombardy and Jacques Sakarovitch. Sequential? *Theor. Comput. Sci.*, 356(1-2):224–244, 2006.

24 M. Lothaire. *Algebraic Combinatorics on Words*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2002.

25 Dana S. Scott. Some definitional suggestions for automata theory. *J. Comput. Syst. Sci.*, 1(2):187–212, 1967.

26 Helmut Seidl. When is a functional tree transduction deterministic? In *TAPSOFT'93: Theory and Practice of Software Development, International Joint Conference CAAP/FASE, Orsay, France, April 13-17, 1993, Proceedings*, volume 668 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 1993.

27 Andreas Weber and Reinhard Klemm. Economy of description for single-valued transducers. *Inf. Comput.*, 118(2):327–340, 1995.