AIX-MARSEILLE UNIVERSITÉ
E.D. 184 MATHEMATIQUES ET INFORMATIQUE
U.F.R. SCIENCES
LABORATOIRE D'INFORMATIQUE ET SYSTÈMES, CNRS U.M.R. 7020

Thèse présentée pour obtenir le grade universitaire de docteur

Discipline : Informatique

Didier VILLEVALOIS

Thesis: Simplifying Transducers using Sequentiality

Titre de la thèse : Simplification de transducteurs en utilisant la séquentialité

Soutenue le 26/11/2019 devant le jury composé de :

Olivier CARTON	IRIF, CNRS, Université Paris-Diderot	Rapporteur
Anca MUSCHOLL	LaBRI, CNRS, Université de Bordeaux	Rapporteure
Christof LÖDING	RWTH Aachen University	Examinateur
Benjamin MONMEGE	LIS, CNRS, Aix-Marseille Université	Examinateur
Jean-Éric PIN	IRIF, CNRS, Université Paris-Diderot	Examinateur
Pierre-Alain REYNIER	LIS, CNRS, Aix-Marseille Université	Directeur de thèse

Abstract

Synthesis is a field of computer science that consists in generating programs from abstract specifications. Specifications are often described via a logical formalism and programs are obtained as models of transformation. While, in the specifications, it is useful to express properties of the desired programs using some forms of non-determinism, we usually want to avoid it in the outcome of synthesis, for obvious efficiency reasons. Generally speaking, this leads us to the need to simplify the synthesised transformation models, in order to optimise their evaluation or translation into practical applications.

In this thesis, the transformation models we are interested in are expressed as Streaming String Transducers (SST) [AČ10 ; AČ11]. An SST is a deterministic finite-state automaton equipped with a finite count of registers that can be used to construct an output word. These registers can be updated by using register concatenation or by prepending or appending finite words. We are interested in the challenging problem of register minimisation, which consists, given an SST, in computing an equivalent SST with a minimal number of registers. As a first step to support this general model, we constrain how the registers can be operated on : namely, the registers cannot be concatenated one to another.

We present two main contributions. First, we devise a procedure allowing to minimise registers in the class of copyless appending SSTs (in this class, registers can only be appended to). Second, we show, given a copyful concatenation-free SST, how to decide whether there exists an equivalent concatenation-free SST with a single register.

When considering the simplification of Finite-State Transducers (FST), a classical problem is the sequentiality problem [Cho77], which asks whether a given FST admits an equivalent sequential one (that is with a deterministic underlying input automaton). For both our results, the proof techniques generalise the framework created around the sequentiality problem.

Keywords : minimisation, sequentiality, transducers, twinning property

Résumé

La synthèse est un domaine de l'informatique consistant à générer des programmes à partir de spécifications abstraites. Les spécifications sont souvent décrites à l'aide d'un formalisme logique et les programmes sont obtenus sous la forme de modèles de transformation. Alors qu'il est utile de pouvoir exprimer les propriétés des spécifications avec du non-déterminisme, nous souhaitons en général obtenir des modèles déterministes pour des raisons évidentes d'efficacité. Ceci nous amène à vouloir simplifier les modèles synthétisés afin d'optimiser leur évaluation ou leur représentation concrète dans un programme.

Dans cette thèse, les modèles de transformations qui nous intéressent sont exprimés par des Streaming String Transducers (SSTs) [AČ10 ; AČ11]. Un SST est un automate fini déterministe équipé d'un nombre fini de registres qui peuvent être utilisés pour élaborer un mot de sortie. Ces registres peuvent être mis à jour en utilisant la concatenation de registres ou en les préfixant ou suffixant par des mots finis. Nous sommes intéressés par le problème ambitieux de la minimisation de registres, qui consiste, étant donné un SST, à calculer un SST équivalent avec un nombre minimal de registres. Comme première étape à la prise en compte de ce modèle très expressif, nous contraignons la manière dont les registres sont manipulés : ils ne peuvent pas être concaténés les uns aux autres (cette classe est appelée Concatenation-Free SST).

Nous présentons deux contributions principales. Tout d'abord, nous élaborons une procédure permettant de minimiser le nombre de registres dans la classe des Copyless Appending SSTs (dans cette classe, les registres ne peuvent qu'être suffixés par un mot). Ensuite, nous montrons, étant donné un Copyful Concatenation-Free SST, comment décider s'il existe un Concatenation-Free SST équivalent à un seul registre.

Lorsque l'on considère la simplification des Finite-State Transducers (FST), un problème classique est le problème de la séquentialité [Cho77], qui demande si un FST donné admet un FST séquentiel équivalent. Pour nos deux résultats, les techniques de preuves utilisées généralisent le cadre créé autour du problème de séquentialité.

Mots clés : minimisation, séquentialité, transducteurs, propriété de jumelage

Remerciements

Le manuscrit entre vos mains marque le point d'orgue d'une tranche de cinq années de vie. Il contient à la fois les doutes, déceptions, sueurs et douleurs, et les surprises, joies et bonheurs qui ont été les miens durant le cheminement qui a mené à son écriture. Heureusement, j'ai eu de nombreuses personnes autour de moi pour m'accompagner.

Quelle idée de reprendre ses études à 40 ans pour se lancer dans une thèse. On me l'a bien fait remarquer quand cette « lubie » m'a prise. Pourtant deux personnes ont accueilli cette idée avec la plus grande bienveillance. C'est à vous, Liva et Pierre-Alain, que j'adresse mes premiers remerciements. Un grand merci, autant pour m'avoir mis en garde, à juste titre, sur mon possible manque de malléabilité dû à mon âge, et pour avoir cru en moi malgré tout et m'avoir soutenu coûte que coûte.

Un an de Master et quatre ans de thèse, c'est une longue période à devoir me supporter et je te remercie encore Pierre-Alain pour ta patience, notamment face à mes régulières démotivations, ralentis, et autres débrayages... Malgré ma grande gueule, je pense n'avoir jamais vraiment été capable de t'exprimer la gratitude que j'ai pour toi.

Je remercie Anca Muscholl et Olivier Carton d'avoir accepté d'être rapporteur.e.s de ce manuscrit, ainsi que les autres membres du jury, Christof Löding, Benjamin Monmège et Jean-Éric Pin pour le temps qu'ils m'ont consacré.

Je remercie aussi les collègues du Master 2 – Célia, Sébastien, Bastien, Éloi ! – ainsi que les doctorants et post-doctorants – Guillaume, Luc, Laure, Matthieu, Christina, Antoine, Makki, Léo – qui m'ont tous merveilleusement bien accueilli parmi eux. Je suis heureux de vous avoir eu à mes côtés durant ces quelques années et j'ai la chance de pouvoir compter certains d'entre vous parmi mes amis pour encore longtemps ! Un coup de coude aux collègues de bureau : Damien, Florian et Éloi. Éloi, je compte sur toi pour en finir rapidement à ton tour.

Je n'aurai pu survivre sans les bulles d'air que m'ont offert mes amies et amis, danseuses et danseurs de Blues, Forró et autres, militantes et militants de la protection animale, et usagères et usagers du Cabas Vert. Je ne peux pas tous vous nommer mais vous vous reconnaîtrez.

Enfin, je remercie ma famille et mes amis proches, qui n'ont jamais mis en doute mes choix de vie et sans qui je n'aurai pu relever ce nouveau défi.

Contents

Introduction

About Transformations

In our everyday lives, everyone of us interacts with machines, be it at work or at home or in between. We feed information to all kinds of computer systems, ranging from smartwatches to autonomous vehicles, to desktop workstations, and expect processed information in return. Each of these devices acts as some form of data transformation. A transformation takes some input data and returns some output data.

Computer science is the study of both computer hardware and software design. In this discipline, we need means to define and realise these transformations. Computer programs are a way to express those: developers use programming languages to encode what input is expected from the user and how the output is computed from it.

Programming languages often have complex syntaxes and semantics, which make them suitable to express complex behaviours in a succinct way. However, their complexity also makes them hard to reason about and prone to error. History is full of hardware and software problems that have led to financial, or more importantly, human disasters [LT93; Ari96].

A basic way to ensure program safety is testing. We confirm, by repeatedly executing the program with a set of valid and invalid inputs, that it behaves correctly and produces the correct output or error message. Nevertheless, the set of valid and invalid inputs is often infinite and thus the program cannot be fully tested. Also, the (often) informal specifications of systems can't describe precisely all the valid and invalid behaviours.

In order to address these issues, a number of mathematically based techniques have evolved over the years: first, some models to abstract transformations, making them easier to reason about and yet expressive enough to model many complex behaviours; second, some formal methods to guarantee the safety of the models we design. We will now introduce some of them, which are of interest to this thesis.

Formal Methods for Safer Systems

The goal of formal methods is to allow the use of mathematical reasoning on the behaviours of the program being developed.

One typical such formal method is model-checking. First, a description of the system to verify is provided, represented as a mathematical model, along with a specification of the admitted behaviours of the system, often expressed using some logical formalism (for example Linear-time Temporal Logic [Pnu77]). Then, a model-checking algorithm is used to verify that the model indeed satisfies the logical formula of the specification. See [Cla+18] for a general overview of model checking.

Another formal method is model synthesis. Here, only a specification of the system, that can also be expressed using a logical formalism, is used as a driver to directly generate a candidate model for the system. This synthesised model is in turn used to produce part or all of the programming language code of the system.

Realisability (also called uniformisation) [CL11] is a particular case of synthesis. In this approach, the specification is directly given as a (maybe non-deterministic) transformation model that expresses a relation between inputs and outputs of the system. From that relation, is extracted a function that has the same domain and behaves equally on that domain.

Efficient Evaluation of Models

When using synthesis, while we have formal guarantees that the synthesised model describes a system that satisfies the specification, this model may not be optimal in a number of ways, like performance or resource consumption. This leads to the need to simplify the synthesised models. Some usual examples for the simplification of models are, for example, the minimisation of their size or the removal of non-determinism.

In this thesis, we focus on the removal, if ever possible, of non-determinism in models. Whereas non-determinism can be useful to express compact system specifications in a logical formalism, it is notoriously impractical in the synthesised models. In fact, the execution of non-deterministic algorithms can sometimes blow up because we can't ensure any bound on the number of parallel evaluation contexts required to complete the computation. This constitutes the main motivation for our work: removing part or all non-determinism in transformation models to allow for their efficient evaluation.

Models of Transformations

As explained above, programming languages, being Türing-complete, are too expressive and many program properties are thus undecidable. We are in need for abstractions that we can reason about more easily. Finite-state models, such as finite-state automata, are a good start. They enjoy good closure and decidability properties and yet are expressive enough to model many complex behaviours.

We can extend finite-state automata to obtain some models of transformations that are also adequate for formal reasoning. In this thesis, we will use such transformation models obtained from finite-state automata: some that are still finite-state and some others that use registers in a carefully restricted way. We will see that we can draw equivalences between those models.

A first aim of this manuscript is to summarise the equivalences between finite-state machines and classes of machines with registers that interest us. We will also summarise the main known results for the problems of functionality, sequentiality and equivalence on these models.

Finite-State Transducers

Finite-state automata can be viewed as functions from words to boolean values, and thus describe languages. Finite-state transducers [Ber13; Sak09] extend finite-state automata by adding output labels in order to produce a word at each transition. The produced words are concatenated along a run in the transducer. Then, given an input word, the output of the transducer is the set of words produced by the runs induced by this input word. As such, finite-state transducers describe relations from words to words.

A finite-state transducer is functional if the relation it realises is a function. The class of functions realised by functional finite-state transducers is called the class of rational functions [Ber13]. We focus in this thesis on functional transducers.

Sequentiality of Finite-State Transducers

A finite-state transducer is sequential (some authors say input-deterministic) if its underlying input automaton is deterministic. Regarding efficient evaluation, sequential finite-state transducers are the silver bullet. Indeed, being sequential, only one evaluation context is required. One can even take the input as a stream of letters and stream the output word in return, thus occupying a constant memory space.

However, not every finite-state transducer can be made into a sequential one. The problem of the sequentiality of finite-state transducers then consists in decid-

ing whether a finite-state transducer admits an equivalent sequential one. This problem has been extensively studied [Cho77; WK94; Béa+00; BC02].

A second aim of this manuscript is to provide a self-contained presentation of known results on sequentiality of finite-state transducers and an original presentation of the decision of the sequentiality problem. Both will serve as a basis for the understanding of our main contributions.

Deterministic Streaming String Transducers

[AČ10; AČ11] recently introduced the model of deterministic streaming string transducers. They are deterministic finite-state automata equipped with registers to store intermediate output words. These registers can be concatenated together or with word constants. A transition in a deterministic streaming string transducer then boils down to a register update.

Register Complexity of Deterministic Streaming String Transducers

A deterministic streaming string transducer is sequential by nature. One way to simplify a deterministic streaming string transducer would be to minimise its number of registers. This is known as the register minimisation problem.

However challenging, a first step has been taken in [DRT16] where this problem is solved for the subclass of copyful appending deterministic streaming string transducers. An appending deterministic streaming string transducer forbids the concatenation of two registers and only allows registers to be appended with some words constants.

When translating between the model of copyful appending deterministic streaming string transducers and the model of functional finite-state transducers, we can observe that the use of registers in one translates in non-determinism in the other. In [DRT16], the authors gave a characterisation of the copyful appending deterministic streaming string transducers that admit an equivalent one with only k registers, for a given $k \in \mathbb{N}$. This work demonstrated that there is a strong connection between register minimisation and sequentialisation. Indeed, when $k = 1$, the register minimisation problem of copyful appending deterministic streaming string transducers is equivalent to the sequentiality problem of finite-state transducers.

It is worth noticing that the class of functions realised by copyful appending deterministic streaming string transducers coincides with the class of rational functions [Alu+13].

This strong connection between the different models of transformations is an invitation to study further the links between sequentiality and register minimisation.

A Family of Models and Sequentiality Problems

k-Sequentiality of Functional Finite-State Transducers

A finite-state transducer is k-sequential if it is the union of k sequential finite-state transducers. It is multi-sequential if it is k-sequential, for some $k \in \mathbb{N}$. As not all finite-state transducers can be made sequential, multi-sequential finite-state transducers appear to be a good compromise in terms of efficient evaluation. Given a functional multi-sequential finite-state transducer, we argue that one can evaluate each member of the union in a separate thread. No communication is required between threads and a single final join is needed in order to collect the output of the only accepting member.

Characterising the finite-state transducers that admit an equivalent multi-sequential one, has been studied in [CS86] for the functional case and in [JF15] for the relational case. The ensuing simplification task would then be to reduce the size of the union. Following in their footsteps, we studied in [Dav+17] the problem of the k-sequentiality of functional finite-state transducers, which is to decide, given $k \in \mathbb{N}$, whether a functional finite-state transducer admits an equivalent k-sequential one.

> The first main contribution of this manuscript is to devise an effective characterisation of the functional finite-state transducers that admit an equivalent k-sequential one, for a given $k \in \mathbb{N}$. We will present this result directly in the setting of finite-state transducers, contrarily to the initial publication, but with some original and more direct proofs.

A copyless deterministic streaming string transducer only allows for registers to be used once in a whole register update of a transition, whereas a copyful deterministic streaming string transducer does not have this restriction. Whereas functional finite-state transducers are equivalent to copyful appending deterministic streaming string transducers, functional k-sequential finite-state transducers are equivalent to copyless appending deterministic streaming string transducers with k registers. This means that the k-sequentiality problem of functional finite-state transducers is equivalent to the problem of the minimisation to k registers of copyless appending deterministic streaming string transducers.

Sequentiality of Functional String-to-Context Transducers

It is tempting to generalise the link between sequentiality and register minimisation further above the rational functions. As functional finite-state transducers are equivalent to copyful appending deterministic streaming string transducers, the next step would be to handle copyful concatenation-free deterministic

streaming string transducers. But there is no equivalent class on the side of finite-state models.

This leads us to define a new model: the string-to-context transducers. Finite-state transducers were able to append to the already produced word (just like copyful appending deterministic streaming string transducers append to their registers). Now, string-to-context transducers can, at each transition, simultaneously prepend and append to the already produced word (just like copyful concatenation-free deterministic streaming string transducers both prepend and append to their registers).

The minimisation to 1 register of copyful appending deterministic streaming string transducers did correspond to the sequentiality of functional finite-state transducers [DRT16]. Being able to decide the sequentiality problem for functional string-to-context transducers would then establish a base case for a future register minimisation of copyful concatenation-free deterministic streaming string transducers.

The second main contribution of this manuscript is to devise an effective characterisation of the functional string-to-context transducers that admit an equivalent sequential one. This constitutes an important first step towards the register minimisation of copyfull concatenation-free deterministic streaming string transducers.

General Methodology

Throughout this manuscript, we will extensively use two main tools:
— properties to characterise classes of functions, mathematically describing in what way their outputs are tied to their inputs, and
— properties to characterise classes of machines, describing a structural pattern that these machines must adhere to.

It is interesting to note that all the main results developed in this manuscript follow the same principles. They are based on the general shape of the sequentialisation theorem of Choffrut [Cho77], that we will recall later. For $\mathcal{C}_1$ a class of machines, and $\mathcal{C}_2$ the subclass of $\mathcal{C}_1$ that we wish to characterise, the main theorem will ressemble to:

Theorem. *Let $\mathcal{T}$ a machine of class $\mathcal{C}_1$. The following assertions are equivalent:*

1. *the function realised by $\mathcal{T}$ satisfies the ... property,*
2. *the machine $\mathcal{T}$ satisfies the ... structural property,*
3. *there exists a machine of class $\mathcal{C}_2$ equivalent to $\mathcal{T}$.*

Each one of theses assertions plays a different role: 1 is a machine independent characterisation, 2 is a property expressed by means of a pattern on a machine of class C_1, that can be used to derive efficient decision procedures, and 3 denotes the class $\mathcal{C}_2$ of machines we wish to characterise.

We will often use a similar overall proof strategy, even if the individual proofs might employ different techniques. This main theorem will always be accompanied by a construction for the $\mathcal{C}_2$ equivalent machine, used as part of the proof for the theorem, and also a decision procedure to identify the machines of $\mathcal{C}_1$ that we characterise.

Outline of this Thesis

This thesis is divided in five chapters.

We start in **Chapter 1** by presenting the different models of transducers that we will use. We also recall the main results for the classical problems of functionality, sequentiality and equivalence for these models.

In **Chapter 2**, we carry out a systematic comparison of the expressiveness of the various equivalent models, by exhibiting constructions from one to another.

Chapter 3 is devoted to the presentation of the main known results around the sequentiality of functional finite-state transducers. We recall the two known properties to characterise the functions realised by finite-state transducers that are sequential or admit a sequential equivalent, along with two different presentations of the structural property to characterise the corresponding finite-state transducers themselves. We then recall the construction of the sequential equivalent when it exists. Finally, we discuss the decision of the sequentiality problem and give a new presentation of the decision procedure.

In **Chapter 4**, we characterise the functional finite-state transducers that admit a k-sequential equivalent. We present a property of the functions that are realised by a k-sequential finite-state transducer and the corresponding structural property for the functional finite-state transducers that admit a k-sequential equivalent. From this, we draw a construction as well as a decision procedure for the k-sequentiality problem.

Chapter 5 presents our work around string-to-context transducers. We first devise a property to characterise the functions that are realised by a sequential string-to-context transducer, along with a structural property to characterise the functional string-to-context transducers that admit a sequential equivalent. We then undertake the combinatorial analysis of those transducers that admit a sequential equivalent, to obtain a more combinatorial version of the structural property. This combinatorial property is later used as a foundation for our construction to build an equivalent sequential string-to-context transducer, if it exists. Finally, we exhibit a decision procedure for the sequentiality problem of string-to-context transducers.

Chapter 1

Models of Transducers

1.1 Preliminaries

1.1.1 Alphabets and Words

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. We denote the *concatenation* of a word u and a word v by $u \cdot v$, and sometimes omit the dot, as in uv, when the meaning is clear from context. The *length* of a word u is denoted by $|u|$. The number of occurrences of the letter a in a word u is denoted by $|u|_{\mathsf{a}}$. We denote by $\mathsf{last}(u)$ the last letter a non-empty word u. We denote by $\tilde{u}$ the *mirror* of the word u, *i.e.* for $u = u_1 \cdots u_k$, where $u_i \in A$ for all $1 \leqslant i \leqslant k$, $\tilde{u} = u_k \cdots u_1$

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 by $\mathsf{root}(u)$, 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 $u \in A^*$, we say that v is a *factor* of u if there exist words x, y such that $u = xvy$.

Given an alphabet B, we denote by $\mathcal{F}_B$ the free group over B. For a word $x \in \mathcal{F}_B$, we denote by x^{-1} its inverse in $\mathcal{F}_B$. For two words $x, y \in B^*$, if x is a prefix of y, resp. a suffix, then we write $x^{-1}y$, resp. yx^{-1}, for the unique word $z \in B^*$ such that $y = xz$, resp. $y = zx$.

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 $\mathsf{dist}_p(u, v)$, as $|u| + |v| - 2|\mathsf{lcp}(u, v)|$.

Given a set of words $W \subseteq A^*$, the longest common prefix (resp. suffix) of words in W is denoted by $\mathsf{lcp}(W)$ (resp. $\mathsf{lcs}(W)$).

In the following lemma, we recall a classical result of combinatorics that we will use in this thesis. For two integers $n > 0$ and $m > 0$, we denote by $\gcd(n, m)$ the greatest common divisor of n and m.

Lemma 1.1 (Fine and Wilf, [FW65], Chapter 9 of [Lot02]). *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 the primitive roots of x and y are conjugates.*

1.1.2 Contexts

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 by $\mathcal{C}(B)$. 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 as $|c| = |\overleftarrow{c}| + |\overrightarrow{c}|$. The *lateralized length* of a context c is defined as $\|c\| = (|\overleftarrow{c}|, |\overrightarrow{c}|)$. 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.

Example 1.1. Let $B = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. Let $c_1 = (\mathsf{aa}, \mathsf{bb}), c_2 = (\mathsf{bc}, \mathsf{c}) \in \mathcal{C}(B)$ and $w = \mathsf{cc} \in B^*$. Then $\overleftarrow{c_1} = \mathsf{aa}$, $\overrightarrow{c_1} = \mathsf{bb}$ and $c_1[w] = \mathsf{aaccbb}$. Also $|c_1| = 4$, $|c_2| = 3$ $\|c_1\| = (2, 2)$ and $\|c_2\| = (2, 1)$. Finally we have $c_1 c_2 = (\mathsf{aabc}, \mathsf{cbb})$.

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 $C.\mathsf{lcc}(C)^{-1} = \{c' \mid c'.\mathsf{lcc}(C) \in C\}$.

Example 1.2. Let $B = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. Let $c_1 = (\mathsf{abab}, \mathsf{bbba})$, $c_2 = (\mathsf{acabab}, \mathsf{bbbb})$, $c_3 = (\mathsf{ccabab}, \mathsf{bbbc}) \in \mathcal{C}(B)$, and let $C = \{c_1, c_2, c_3\}$. Then $\mathsf{lcc}(C) = (\mathsf{abab}, \mathsf{bbb})$. And $C.\mathsf{lcc}(C)^{-1} = \{(\varepsilon, \mathsf{a}), (\mathsf{ac}, \mathsf{b}), (\mathsf{cc}, \mathsf{c})\}$

1.1.3 Functions and Relations

We consider two sets X, Y. Given a (binary) relation $\Delta \subseteq X \times Y$, we let $\mathsf{dom}(\Delta) = \{x \in X \mid \exists y \in Y \text{ such that } (x, y) \in \Delta\}$. We denote the set of partial functions from X to Y by $\mathcal{F}(X, Y)$. Given $f \in \mathcal{F}(X, Y)$, we write $f : X \hookrightarrow Y$, we denote by $\mathsf{dom}(f)$ its domain and, for $X' \subseteq X$, we define $f(X') = \{f(x) \mid x \in \mathsf{dom}(f) \cap X'\}$. 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 $\mathsf{dom}(\Delta) = \mathsf{dom}(\Delta')$.

Remark. Throughout this thesis, we consider that all the functions we manipulate, be it the ones that we characterise or the ones that our models realise, are potentially partial. Therefore, we generally omit to say that they are indeed partial. Sometimes, however, we will explicitly insist that a function is partial if this is important for the understanding of the matter at hand.

1.1.4 Models and Associated Problems

In this thesis, we will manipulate transducers that realise functions from words to words or (binary) relations between words. We call *class of transducers* a set of transducers that either are described using the same model or satisfy some common properties.

In our domain, we often study whether (and how) some general problems can be solved with respect to a particular class of machines. We now introduce some classical problems of interest.

We consider a class $\mathcal{C}$ of transducers. The following problem asks, given a transducer in $\mathcal{C}$, whether this transducer is a member of the subclass of $\mathcal{C}$ consisting only of transducers realising functions.

Problem 1.1 (Functionality)**.** Given a transducer of class $\mathcal{C}$, decide whether it realises a function.

The models of transducers we are interested in are all extensions of finite-state automata on words. From a transducer considered here, we can always extract the underlying automaton. If this underlying automaton is deterministic, then the transducer is said to be *sequential*. This leads to the following problem which, given a transducer in $\mathcal{C}$, asks whether this transducer admits an equivalent transducer in the subclass of $\mathcal{C}$ consisting only of sequential transducers.

Problem 1.2 (Sequentiality)**.** Given a transducer of class $\mathcal{C}$, decide whether it can be realised by an equivalent sequential transducer of class $\mathcal{C}$.

The following problem is also a natural problem and asks, given two transducers in $\mathcal{C}$, whether these transducers are equivalent.

Problem 1.3 (Equivalence)**.** Given two transducers of class $\mathcal{C}$, decide whether they realise the same function or relation.

For each of the class of transducers that we introduce hereafter, we will recall the existing results for these classical problems. All these results are summarised in Table 1.1.

1.2 Finite-State Transducers

Like finite-state automata, *finite-state transducers* read their input on a one-way left-to-right input tape. In addition, they can write to a one-way left-to-right output tape. For a comprehensive review, see [Ber13] or [Sak09].

Definition 1.1. Let A, B be two finite alphabets. A *finite-state transducer* (FST for short) $\mathcal{T}$ from A^* to B^* is a tuple $(Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ where Q is a finite set of states, $t_{\mathit{init}} : Q \hookrightarrow B^*$ (*resp.* $t_{\mathit{final}} : Q \hookrightarrow B^*$) is the initial (*resp.* final) function, and $T \subseteq Q \times A \times B^* \times Q$ is the finite set of transitions.

A state q is said to be *initial* (*resp. final*) if $q \in \mathsf{dom}(t_{\mathit{init}})$ (*resp.* $q \in \mathsf{dom}(t_{\mathit{final}})$). We depict as $\xrightarrow[\mathcal{T}]{w} q$ (*resp.* $q \xrightarrow[\mathcal{T}]{w}$), or just $\xrightarrow{w} q$ (*resp.* $q \xrightarrow{w}$) if it is clear from the context, the fact that $t_{\mathit{init}}(q) = w$ (*resp.* $t_{\mathit{final}}(q) = w$). A *run* ρ from a state q_1 to a state q_{k+1} on a word $u = u_1 \cdots u_k \in A^*$ where for all $1 \leqslant j \leqslant k$, $u_i \in A$, is a sequence of transitions $(q_1, u_1, w_1, q_2), (q_2, u_2, w_2, q_3), \ldots, (q_k, u_k, w_k, q_{k+1})$. The *output* of such a run is the word $w = w_1 w_2 \ldots w_k \in B^*$, and is denoted by $\mathsf{out}(\rho)$. We depict this situation as $q_1 \xrightarrow[\mathcal{T}]{u|w} q_{k+1}$, or just $q_1 \xrightarrow{u|w} q_{k+1}$ if it is clear from the context. The run ρ is said to be *accepting* if q_1 is initial and q_{k+1} final. This finite-state transducer $\mathcal{T}$ computes a relation $[\![\mathcal{T}]\!] \subseteq A^* \times B^*$ defined as the set of pairs $(u, v_1v_2v_3)$ such that there are $p, q \in Q$ with $\xrightarrow[\mathcal{T}]{v_1} p \xrightarrow[\mathcal{T}]{u|v_2} q \xrightarrow[\mathcal{T}]{v_3}$ an accepting run.

Given an FST $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$, we define the constant $M_{\mathcal{T}}$ as $M_{\mathcal{T}} = \max\{|w| \mid (p, a, w, q) \in T \text{ or } (q, w) \in t_{\mathit{init}} \cup t_{\mathit{final}}\}$. Given $\Delta : Q \hookrightarrow B^*$, we denote by $\mathcal{T}_\Delta$ the FST obtained by replacing t_{init} with Δ.

An FST is *trimmed* if each of its states appears in some accepting run. W.l.o.g., we assume that the finite-state transducers we consider are trimmed. Indeed, we are in this thesis only interested in the accepting runs and an equivalent trimmed FST can be built in linear time in the number of states.

The union of two finite-state transducers $\mathcal{T}_i = (Q_i, t^i_{\mathit{init}}, t^i_{\mathit{final}}, T_i)$, for $i \in \{1, 2\}$, is defined as $\mathcal{T}_1 \cup \mathcal{T}_2 = (Q_1 \cup Q_2, t^1_{\mathit{init}} \cup t^2_{\mathit{init}}, t^1_{\mathit{final}} \cup t^2_{\mathit{final}}, T_1 \cup T_2)$. States can always be renamed to ensure disjointness. It is trivial to verify that $[\![\mathcal{T}_1 \cup \mathcal{T}_2]\!] = [\![\mathcal{T}_1]\!] \cup [\![\mathcal{T}_2]\!]$. This operation can be generalized to the union of k finite-state transducers.

An FST $\mathcal{T}$ from A^* to B^* is *functional* if the relation $[\![\mathcal{T}]\!]$ is a function from A^* to B^*. The class of functions realised by functional finite-state transducers is called the class of *rational functions* [Ber13] and is denoted by Rat.

An FST $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ is *sequential* if $\mathsf{dom}(t_{\mathit{init}})$ is a singleton and if for every transitions $(p, a, w, q), (p, a, w', q') \in T$, we have $q = q'$ and $w = w'$. As it computes a unique run per input word, a sequential FST is always functional. An

FST is k-*sequential* if it is the union of k sequential FSTs. It is *multi-sequential* if it is k-sequential for some $k \in \mathbb{N}$.

Figure 1.1 describes the graphical notations used to depict finite-state transducers throughout this thesis.

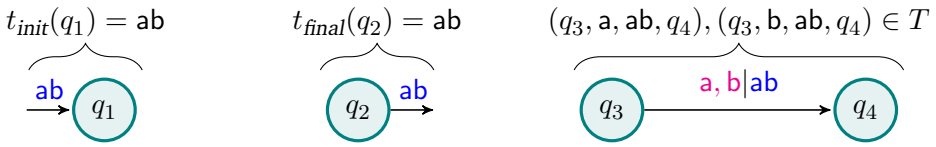


Figure 1.1 – Graphical notations used to depict FSTs.

Example 1.3. Two examples of functional, but non-sequential, finite-state transducers are depicted on Figure 1.2. $\mathcal{T}_{last}$, on Figure 1.2a, computes the function $f_{last} : u \in \{a, b\}^+ \mapsto \mathsf{last}(u)^{|u|}$. $\mathcal{T}_{last}$ first non-deterministically guesses the last letter of the input word, and hence decides which of the initial states q_a or q_b to start from. While in q_a or q_b, it also guesses whether the current input letter is the last letter of the input word, and hence whether it should move to the final state q_f. $\mathcal{T}_{last^*}$, on Figure 1.2b, computes the function $f_{last^*} : u_1 \# \cdots \# u_n \mapsto \mathsf{last}(u_1)^{|u_1|} \# \cdots \# \mathsf{last}(u_n)^{|u_n|}$ where for all $1 \leqslant i \leqslant n$, $u_i \in \{a, b\}^+$. $\mathcal{T}_{last^*}$ operates similarly to $\mathcal{T}_{last}$ and makes the same non-deterministic guesses but for the current $\#$-separated input subword.

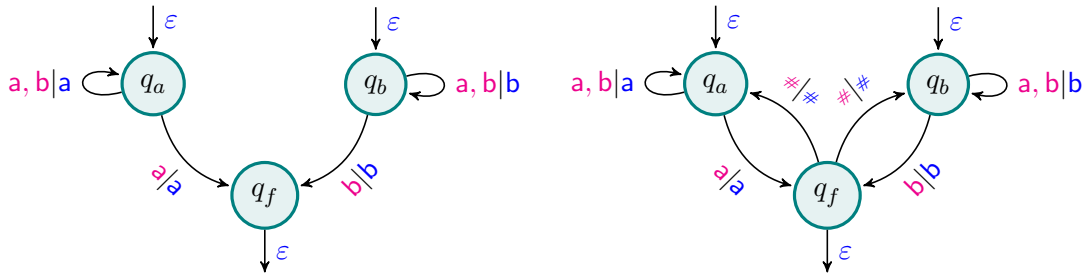


Figure 1.2 – Two example FSTs: $\mathcal{T}_{last}$ and $\mathcal{T}_{last^*}$.

The problem of the functionality of FSTs has long been studied [Sch75; GI83; Béa+00] and proved by [FMR18] to be NLOGSPACE-complete. The problem of the sequentiality of FSTs has also been studied extensively [Cho77; WK94; Béa+00; BC02] and again proved by [FMR18] to be NLOGSPACE-complete.

The equivalence problem for sequential FSTs and functional FSTs has been studied in [BH79] and [BH77] respectively. Regarding complexity, we can show it is NLOGSPACE-complete for sequential FSTs and PSPACE-complete for functional FSTs. The equivalence problem has been shown to be undecidable for FSTs in general [Iba77].

1.3 Two-way Finite-State Transducers

Two-way finite-state transducers are analogous to finite-state transducers except that they use a two-way input tape, as two-way finite-state automata do [AHU69]. The input word can therefore be traversed in both directions arbitrarily. As such, the transducer needs to know when it reached the boundaries of the input word. Therefore we enrich the input alphabet with some begin and end markers (namely $\vdash$ and $\dashv$) used to flank the input word.

Definition 1.2. Let A, B be two finite alphabets. A *two-way finite-state transducer* (2FST for short) $\mathcal{T}$ from A^* to B^* is a tuple $(Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ where Q is a finite set of states, $t_{\mathit{init}} : Q \hookrightarrow B^*$ (*resp.* $t_{\mathit{final}} : Q \hookrightarrow B^*$) is the initial (*resp.* final) function, and $T \subseteq Q \times A_{\vdash\dashv} \times B^* \times Q \times \{-1, +1\}$ is the finite set of transitions, where $A_{\vdash\dashv} = A \cup \{\vdash, \dashv\}$.

A *configuration* of a 2FST is a pair $(q, i) \in Q \times \mathbb{N}$ where q is a state, and i is the current position on the input tape. A state q is said to be *initial* (*resp. final*) if $q \in \mathsf{dom}(t_{\mathit{init}})$ (*resp.* $q \in \mathsf{dom}(t_{\mathit{final}})$). A *run* ρ from a state q_1 to a state q_{k+1} on a word $u = u_1 \cdots u_n \in A_{\vdash\dashv}^*$ where for all $1 \leqslant i \leqslant n$, $u_i \in A_{\vdash\dashv}$, is a sequence of configurations $(q_1, i_1), (q_2, i_2), \ldots, (q_{k+1}, i_{k+1})$ such that, for all $1 \leqslant j \leqslant k$, we have $1 \leqslant i_j \leqslant n$ and there exists $(q_j, u_{i_j}, w_j, q_{j+1}, m_j) \in T$, such that $m_j = i_{j+1} - i_j$. The *output* of such a run is the word $w = w_1 w_2 \ldots w_k \in B^*$, and is denoted by $\mathsf{out}(\rho)$. The run ρ is said to be *accepting* if q_1 is initial, q_{k+1} is final, $i_1 = 1$ and $i_{k+1} = n + 1$. This two-way finite-state transducer $\mathcal{T}$ computes a relation $[\![\mathcal{T}]\!] \subseteq A^* \times B^*$ defined as the set of pairs $(u, t_{\mathit{init}}(p) \cdot \mathsf{out}(\rho) \cdot t_{\mathit{final}}(q))$ such that there are two states $p, q \in Q$ and an accepting run ρ from p to q on the word $\vdash u \dashv$.

A 2FST $\mathcal{T}$ from A^* to B^* is *functional* if the relation $[\![\mathcal{T}]\!]$ is a function from A^* to B^*. Based on the equivalence of this model with Courcelle's monadic second-order logic definable string transductions [EH01], the class of functions realised by functional two-way finite-state transducers is called the class of *regular functions* and is denoted by Reg.

A 2FST $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ is *sequential* if $\mathsf{dom}(t_{\mathit{init}})$ is a singleton and if for every transitions $(p, a, w, q, m), (p, a, w', q', m') \in T$, we have $q = q'$, $w = w'$ and $m = m'$. As it computes a unique run per input word, a sequential 2FST is always functional.

Figure 1.3 describes the graphical notations used to depict two-way finite-state transducers throughout this thesis.

Example 1.4. Three examples of functional two-way finite-state transducers are depicted on Figure 1.4. $\mathcal{T}_{\mathit{mirror}}$, $\mathcal{T}_{\mathit{partition}}$ and $\mathcal{T}_{\mathit{copy}}$ all operate by making three traversals of the input word. All three of them are sequential. $\mathcal{T}_{\mathit{mirror}}$, on Figure 1.4a, computes the function $f_{\mathit{mirror}} : u \in \{\mathsf{a}, \mathsf{b}\}^* \mapsto \tilde{u}$. $\mathcal{T}_{\mathit{mirror}}$ moves forward

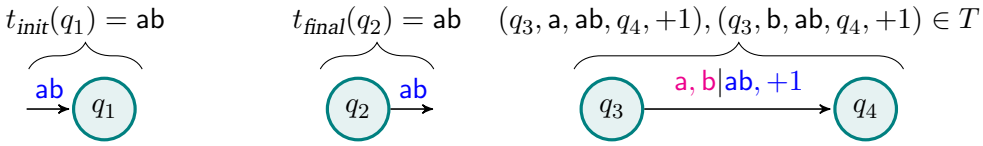


Figure 1.3 – Graphical notations used to depict 2FSTs.

to the end of the input word (in state q_2), copies the letters while moving backward to its beginning (in state q_3), and finally moves forward back to its end (in state q_4). $\mathcal{T}_{partition}$, on Figure 1.4b, computes the function $f_{partition} : u \in \{\mathsf{a}, \mathsf{b}\}^* \mapsto \mathsf{a}^{|u|_\mathsf{a}}\mathsf{b}^{|u|_\mathsf{b}}$. $\mathcal{T}_{partition}$ moves forward while collecting the a letters (in state q_2), moves backward to the beginning of the input word (in state q_3), and finally moves forward again while collecting the b letters (in state q_4). $\mathcal{T}_{copy}$, on Figure 1.4c, computes the function $f_{copy} : u \in \{\mathsf{a}, \mathsf{b}\}^* \mapsto uu$. $\mathcal{T}_{copy}$ moves forward while copying the input word a first time (in state q_2), moves backward to its beginning (in state q_3), and finally moves forward again while copying the input word a second time (in state q_4).

The problem of the functionality of 2FSTs has been shown to be PSPACE-complete [CK87]. [EH01] proved that the class of functional 2FSTs and the class of sequential 2FSTs coincide. As such, functional 2FSTs can always be realised by an equivalent sequential 2FST and [CK87] makes the problem of the sequentiality of 2FSTs to be PSPACE-complete.

The equivalence problem for sequential 2FSTs has been shown to be PSPACE-complete [Gur80]. This complexity also holds for functional 2FSTs as explained in [MP19]. As the problem of the equivalence is undecidable for FSTs in general [Iba77], it is also undecidable for 2FSTs in general.

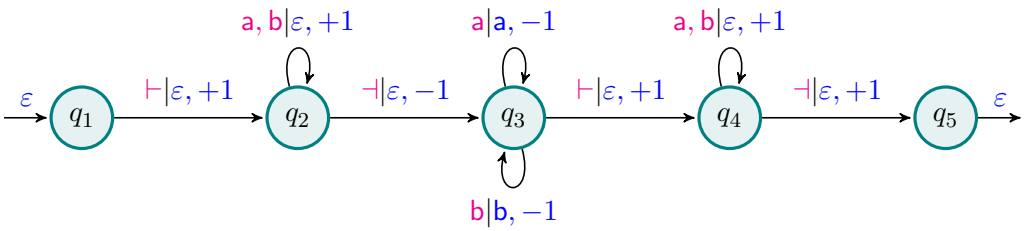
1.4 Deterministic Streaming String Transducers

Alur and Černý recently proposed a new model, *deterministic streaming string transducers*, to capture the class of regular functions [AČ10; AČ11]. Contrarily to finite-state transducers and two-way finite-state transducers, deterministic streaming string transducers are not finite-state. Although they read the input word on a simple one-way left-to-right input tape, they employ a finite number of output registers, that can be concatenated altogether and with finite words, and thus recover a lot of expressiveness.

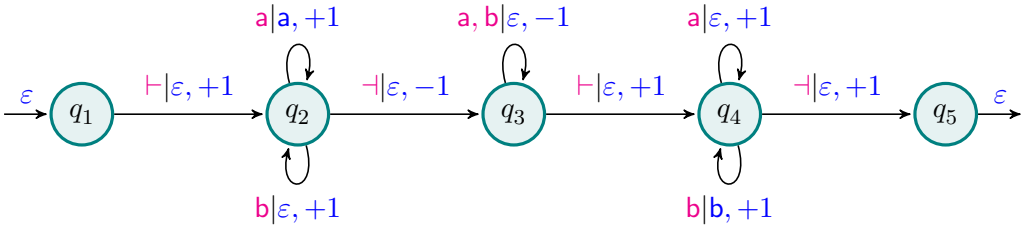
The following three definitions explain how registers are operated on.

Definition 1.3 (Register selectors). Given a finite set of registers $\mathcal{X}$ and an alphabet B, we denote by $\mathcal{S}el(\mathcal{X}, B)$ the set of *register selectors* defined as $(\mathcal{X} \cup B)^*$.

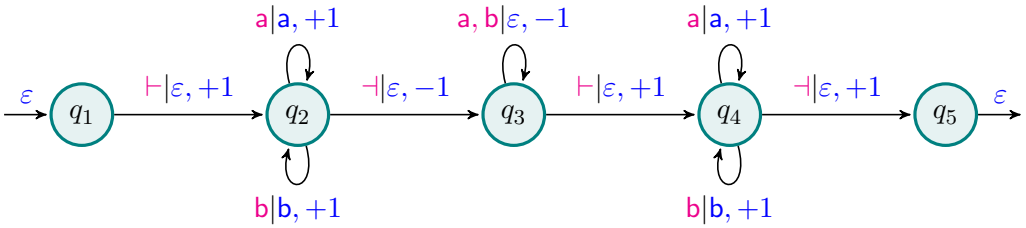
Definition 1.4 (Valuations). Given a finite set of registers $\mathcal{X}$ and an alphabet B, we define *valuations* as mappings from $\mathcal{X}$ to B^*. Let $\mathcal{V}al(\mathcal{X}, B)$ be the set of such



(a) $\mathcal{T}_{mirror}$



(b) $\mathcal{T}_{partition}$



(c) $\mathcal{T}_{copy}$

Figure 1.4 – Three example 2FSTs: $\mathcal{T}_{mirror}$, $\mathcal{T}_{partition}$ and $\mathcal{T}_{copy}$.

valuations. For $\nu \in \mathcal{V}al(\mathcal{X}, B)$, we lift ν to a morphism on $\mathcal{S}el(\mathcal{X}, B)$ by setting $\nu(a) = a$ for all $a \in B$.

Definition 1.5 (Update functions)**.** Given a finite set of registers $\mathcal{X}$ and an alphabet B, the set of *update functions* $\mathcal{X} \to \mathcal{S}el(\mathcal{X}, B)$ is denoted by $\mathcal{U}pd(\mathcal{X}, B)$. For $\sigma \in \mathcal{U}pd(\mathcal{X}, B)$, we lift σ to a morphism on $\mathcal{S}el(\mathcal{X}, B)$ by setting $\sigma(a) = a$ for all $a \in B$. For $\nu \in \mathcal{V}al(\mathcal{X}, B)$ and $\sigma \in \mathcal{U}pd(\mathcal{X}, B)$, we write $\sigma(\nu) \in \mathcal{V}al(\mathcal{X}, B)$ to denote the application of σ to ν, where for all $X \in \mathcal{X}$, $\sigma(\nu)(X) = \nu(\sigma(X))$. Finally, for $\sigma_1, \sigma_2 \in \mathcal{U}pd(\mathcal{X}, B)$, we write $\sigma_1\sigma_2 \in \mathcal{U}pd(\mathcal{X}, B)$ to denote the composition of σ_1 and σ_2, where for all $X \in \mathcal{X}$, $\sigma_1\sigma_2(X) = \sigma_2(\sigma_1(X))$.

For the purpose of depicting update functions, we use sets of assignments of the form "$X = s$" where $X \in \mathcal{X}$ and $s \in \mathcal{S}el(\mathcal{X}, B)$.

Example 1.5. Let $\mathcal{X} = \{X, Y\}$ and $B = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. Let $\nu = \{X \mapsto \mathsf{ab}; Y \mapsto \varepsilon\}$ be a valuation. Let $\sigma = \{X = XY; Y = X\mathsf{c}\}$ be an update function. Then

we have $\sigma(\nu) = \{X \mapsto \mathsf{ab}; Y \mapsto \mathsf{abc}\}$, $\sigma\sigma(\nu) = \{X \mapsto \mathsf{ababc}; Y \mapsto \mathsf{abc}\}$, and $\sigma\sigma\sigma(\nu) = \{X \mapsto \mathsf{ababcabc}; Y \mapsto \mathsf{ababcc}\}$.

Definition 1.6. Let A, B be two finite alphabets. A *deterministic streaming string transducer* (DSST for short) $\mathcal{T}$ from A^* to B^* is a tuple $(Q, \mathcal{X}, q_{init}, \nu, \delta, \mu)$ where Q is a finite set of states, $\mathcal{X}$ is a finite set of registers, $q_{init} \in Q$ is the initial state, $\nu \in \mathcal{V}al(\mathcal{X}, B)$ is the initial valuation, $\delta : Q \times A \hookrightarrow \mathcal{U}pd(\mathcal{X}, B) \times Q$ is the transition function, $\mu : Q \hookrightarrow \mathcal{S}el(\mathcal{X}, B)$ is the final selection function.

Remark. Note that in most presentations of this model, as for example in [AČ10], the initial valuation always associates every registers to the empty word. However, we choose here to have an explicit initial valuation, as it will simplify the presentation of our results. It can easily be shown that both ways are equivalent.

We depict as $\xrightarrow[\mathcal{T}]{\nu} q_{init}$, or just $\xrightarrow{\nu} q_{init}$ if it is clear from the context, the fact that q_{init} is the initial state and ν the initial valuation. A state q is said to be *final* if $q \in \mathsf{dom}(\mu)$, and we depict as $q \xrightarrow[\mathcal{T}]{s}$, or just $q \xrightarrow{s}$ if it is clear from the context, the fact that $\mu(q) = s$. A *run* ρ from a state q_1 to a state q_{k+1} on a word $u = u_1 \cdots u_k \in A^*$ where for all $1 \leqslant i \leqslant k$, $u_i \in A$, is a sequence of transitions $(q_1, u_1, \sigma_1, q_2), (q_2, u_2, \sigma_2, q_3), \ldots, (q_k, u_k, \sigma_k, q_{k+1})$ such that for all i, $\delta(q_i, u_i) = (\sigma_i, q_{i+1})$. The *update* of such a run is the update function $\sigma = \sigma_1\sigma_2 \cdots \sigma_k \in \mathcal{U}pd(\mathcal{X}, B)$. We depict this situation as $q_1 \xrightarrow[\mathcal{T}]{u|\sigma} q_{k+1}$ or just $q_1 \xrightarrow{u|\sigma} q_{k+1}$ if it is clear from the context. The run ρ is said to be *accepting* if q_1 is initial and q_{k+1} final. This deterministic streaming string transducer $\mathcal{T}$ computes a function [1] $[\![\mathcal{T}]\!] : A^* \hookrightarrow B^*$ such that $[\![\mathcal{T}]\!](u) = \sigma(\nu)(s)$, for all $u \in A^*$, for which there are $p, q \in Q$ with $\xrightarrow[\mathcal{T}]{\nu} p \xrightarrow[\mathcal{T}]{u|\sigma} q \xrightarrow[\mathcal{T}]{s}$ an accepting run.

A DSST is *trimmed* if each of its states appears in some accepting run. W.l.o.g., we assume that the deterministic streaming string transducers we consider are trimmed. Indeed, we are in this thesis only interested in the accepting runs and an equivalent trimmed DSST can be built in linear time in the number of states.

Definition 1.7 (Copyless/Copyful)**.** We say that a register selector $s \in \mathcal{S}el(\mathcal{X}, B)$ is *copyless* if each $X \in \mathcal{X}$ occurs at most once in s. An update function σ is *copyless* if each $X \in \mathcal{X}$ occurs at most once in $\sigma(\mathcal{X})$. A DSST is *copyless* if all its transitions are labelled with copyless update functions or register selectors. If a DSST is not copyless, then it is *copyful*.

Remark. The copyless restriction is also called *linear* by some authors. As a matter of fact, it is very similar to the linear constraint of tree transducers [Com+07].

Figure 1.5 describes the graphical notations used to depict deterministic streaming string transducers throughout this thesis.

1. Observe that, by definition, DSSTs always define functions.

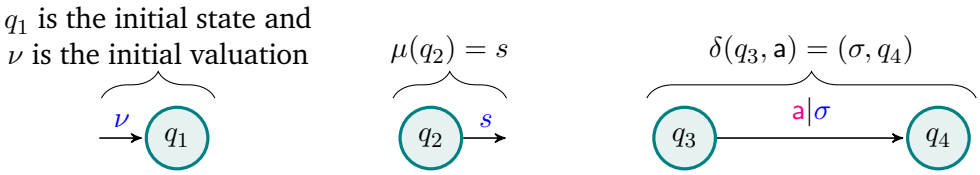
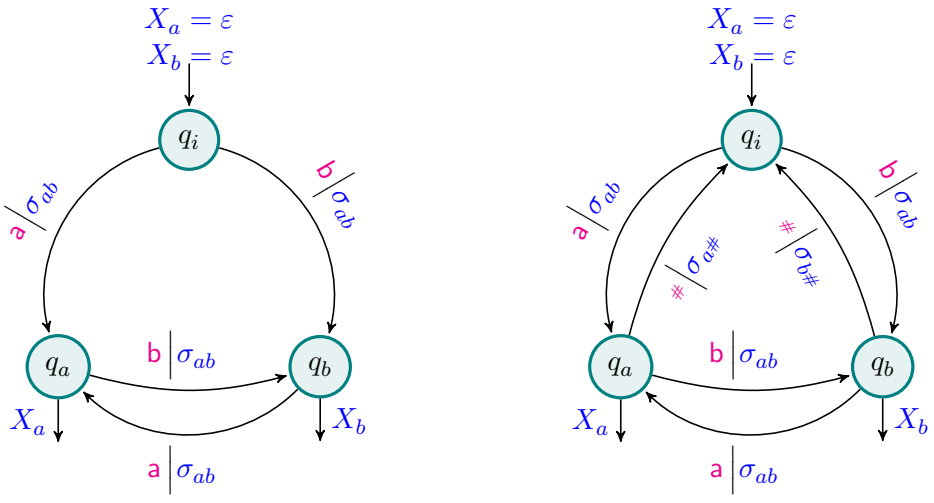


Figure 1.5 – Graphical notations used to depict DSSTs.

Example 1.6. Two examples of deterministic streaming string transducers are depicted on Figure 1.6. $\mathcal{S}_{last}$ and $\mathcal{S}_{last^*}$ are equivalent to $\mathcal{T}_{last}$ and $\mathcal{T}_{last^*}$ of Example 1.3.

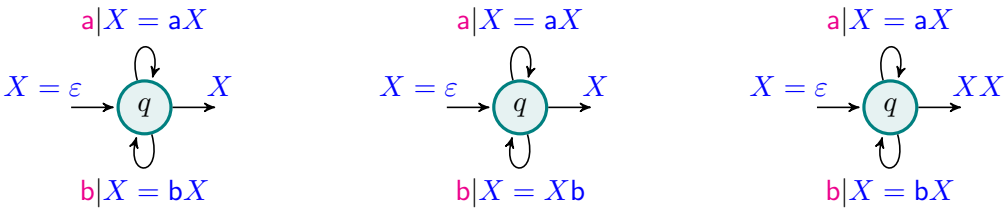
$\mathcal{S}_{last}$, on Figure 1.6a, computes the function $f_{last} : u \in \{a, b\}^+ \mapsto \mathsf{last}(u)^{|u|}$. $\mathcal{S}_{last}$ replaces non-determinism by the use of two registers X_a and X_b to store both the words that would be produced if the last input letter were to be an a or a b. It also remembers in states q_a and q_b whether the last read letter is an a or a b, and selects the corresponding register X_a or X_b if this is indeed the last input letter. $\mathcal{S}_{last^*}$, on Figure 1.6b, computes the function $f_{last^*} : u_1\# \cdots \# u_n \mapsto \mathsf{last}(u_1)^{|u_1|}{}_\# \cdots {}_\# \mathsf{last}(u_n)^{|u_n|}$ where for all $1 \leqslant i \leqslant n$, $u_i \in \{a, b\}^+$. $\mathcal{S}_{last^*}$ operates similarly to $\mathcal{S}_{last}$. Additionally, when it reads a $\#$ from states q_a and q_b, it copies the content of the register X_a or X_b to the other register in order to commit to the word produced accordingly to the last letter. As a result, X_a and X_b always have equal content up to their last $\#$ letter. Note that whereas $\mathcal{S}_{last}$ is copyless, $\mathcal{S}_{last^*}$ is copyful, because of the parallel copies of the register X_a (*resp.* X_b) in the update function $\sigma_{a\#}$ (*resp.* $\sigma_{b\#}$).



(a) $\mathcal{S}_{last}$ *(b)* $\mathcal{S}_{last^*}$

Figure 1.6 – Two example DSSTs: $\mathcal{S}_{last}$ and $\mathcal{S}_{last^*}$. The updates are abbreviated: $\sigma_{ab} = \{X_a = X_a\mathsf{a}; X_b = X_b\mathsf{b}\}$, $\sigma_{a\#} = \{X_a = X_a\#; X_b = X_a\#\}$ and $\sigma_{b\#} = \{X_a = X_b\#; X_b = X_b\#\}$.

Example 1.7. Three additional examples of deterministic streaming string transducers are depicted on Figure 1.7. $\mathcal{S}_{mirror}$, $\mathcal{S}_{partition}$ and $\mathcal{S}_{copy}$ are equivalent to $\mathcal{T}_{mirror}$, $\mathcal{T}_{partition}$ and $\mathcal{T}_{copy}$ of Example 1.4. They all replace the two-way traversals by the use of a unique register X. $\mathcal{S}_{mirror}$, on Figure 1.7a, computes the function $f_{mirror} : u \in \{\mathsf{a},\mathsf{b}\}^* \mapsto \tilde{u}$. $\mathcal{S}_{mirror}$ prepends the read letters to X, thus obtaining the mirror of the input word. $\mathcal{S}_{partition}$, on Figure 1.7b, computes the function $f_{partition} : u \in \{\mathsf{a},\mathsf{b}\}^* \mapsto \mathsf{a}^{|u|_\mathsf{a}}\mathsf{b}^{|u|_\mathsf{b}}$. $\mathcal{S}_{partition}$ prepends the read a letters and appends the read b letters, thus obtaining the partitioning of those letters. $\mathcal{S}_{copy}$, on Figure 1.7c, computes the function $f_{copy} : u \in \{\mathsf{a},\mathsf{b}\}^* \mapsto uu$. $\mathcal{S}_{copy}$ appends the read letters to X and finally uses two copies of X.



(a) $\mathcal{S}_{mirror}$ (b) $\mathcal{S}_{partition}$ (c) $\mathcal{S}_{copy}$

Figure 1.7 – Three example DSSTs: $\mathcal{S}_{mirror}$, $\mathcal{S}_{partition}$ and $\mathcal{S}_{copy}$.

Definition 1.8 (Appending/Concatenation-free)**.** We define two subsets of $\mathcal{Sel}(\mathcal{X}, B)$: $\mathcal{Sel}_{app}(\mathcal{X}, B) = \mathcal{X} \cdot B^*$ and $\mathcal{Sel}_{cf}(\mathcal{X}, B) = B^* \cdot \mathcal{X} \cdot B^*$. We say that a register selector s is *appending* (*resp. concatenation-free*) if $s \in \mathcal{Sel}_{app}(\mathcal{X}, B)$ (*resp.* $s \in \mathcal{Sel}_{cf}(\mathcal{X}, B)$). An update function σ is *appending* (*resp. concatenation-free*) if, for all $X \in \mathcal{X}$, $\sigma(X) \in \mathcal{Sel}_{app}(\mathcal{X}, B)$ (*resp.* $\sigma(X) \in \mathcal{Sel}_{cf}(\mathcal{X}, B)$). A DSST is *appending* (*resp. concatenation-free*) if all its transitions are labelled with *appending* (*resp. concatenation-free*) update functions or register selectors.

Example 1.8. $\mathcal{S}_{last}$ and $\mathcal{S}_{last^*}$ are appending, and thus concatenation-free. $\mathcal{S}_{mirror}$ and $\mathcal{S}_{part}$ are concatenation-free but not appending. $\mathcal{S}_{copy}$ is neither appending nor concatenation-free.

The equivalence problem for copyless DSSTs has been shown to be in PSPACE [AČ11]. It has also been shown to be decidable for copyful DSSTs [FR17].

1.5 String-to-Context Transducers

In this section, we introduce the *string-to-context transducers*, that are the basis for some of the work presented in this thesis. They stand as an alternative presentation of the (copyless) concatenation-free DSSTs with 1 register. Unlike two-way finite-state transducers, that extend the finite-state transducers with a

two-way input tape, string-to-context transducers keep a one-way input tape but extend their output tape. The output tape can now be written to at both ends simultaneously, *i.e.* the output word is both prepended and appended to at each transition.

Definition 1.9. 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 initial (*resp.* final) function, and $T \subseteq Q \times A \times \mathcal{C}(B) \times Q$ 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 $\xrightarrow[\mathcal{T}]{c} q$ (*resp.* $q \xrightarrow[\mathcal{T}]{c}$), or just $\xrightarrow{c} q$ (*resp.* $q \xrightarrow{c}$) if it is clear from the context, 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+1} on a word $u = u_1 \cdots u_k \in A^*$ where for all $1 \leqslant j \leqslant k$, $u_i \in A$, is a sequence of transitions $(q_1, u_1, c_1, q_2), (q_2, u_2, c_2, q_3), \ldots, (q_k, u_k, c_k, q_{k+1})$. The *output* of such a run is the context $c = c_k c_{k-1} \cdots c_1 \in \mathcal{C}(B)$, and is denoted by $\mathsf{out}(\rho)$. We depict this situation as $q_1 \xrightarrow[\mathcal{T}]{u|c} q_{k+1}$, or just $q_1 \xrightarrow{u|c} q_{k+1}$ if it is clear from the context. 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 as the set of pairs $(u, d_3 d_2 d_1[\varepsilon])$ such that there are $p, q \in Q$ with $\xrightarrow[\mathcal{T}]{d_1} p \xrightarrow[\mathcal{T}]{u|d_2} q \xrightarrow[\mathcal{T}]{d_3}$ an accepting run.

Remark. Because of the insertion of an empty word inside the context produced by runs, the output of S2Cs are words and not contexts. This effectively makes S2Cs string-to-string transducers.

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 $\Delta : Q \hookrightarrow B^*$, 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. Indeed, we are in this thesis only interested in the accepting runs and an equivalent trimmed S2C can be built in linear time in the number of states.

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 $c = c'$. As it computes a unique run per input word, a sequential S2C is always functional.

Figure 1.8 describes the graphical notations used to depict string-to-context transducers throughout this thesis.

Example 1.9. Two examples of string-to-context transducers are depicted on Figure 1.9. $\mathcal{T}'_{mirror}$ and $\mathcal{T}'_{partition}$ are equivalent to $\mathcal{T}_{mirror}$ and $\mathcal{T}_{partition}$ of Example 1.4. They both are sequential and replace the two-way traversals by the use of prepending. $\mathcal{T}'_{mirror}$, on Figure 1.9a, computes the function $f_{mirror} : u \in \{\mathsf{a}, \mathsf{b}\}^* \mapsto$

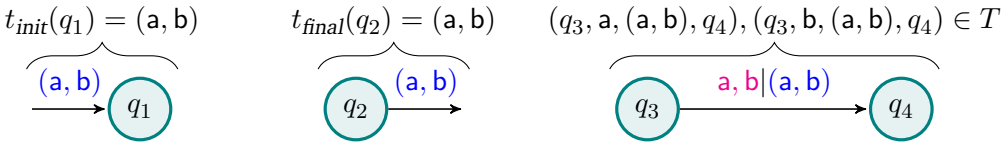


Figure 1.8 – Graphical notations used to depict S2Cs.

$\tilde{u}$. $\mathcal{T}'_{mirror}$ prepends the read letters, thus obtaining the mirror of the input word. $\mathcal{T}'_{partition}$, on Figure 1.9b, computes the function $f_{partition} : u \in \{\mathsf{a}, \mathsf{b}\}^* \mapsto \mathsf{a}^{|u|_\mathsf{a}}\mathsf{b}^{|u|_\mathsf{b}}$. $\mathcal{T}'_{partition}$ prepends the read a letters and appends the read b letters, thus obtaining the partitioning of those letters.

(a) $\mathcal{T}'_{mirror}$ *(b)* $\mathcal{T}'_{partition}$

Figure 1.9 – Two example S2Cs: $\mathcal{T}'_{mirror}$ and $\mathcal{T}'_{partition}$.

Remark. The classical model of finite-state transducers can be recovered from the one of string-to-context transducers. A string-to-context transducer $\mathcal{T} = (Q, t_{init}, t_{final}, T)$ is a string-to-string transducer from A^* to B^* if, for all $(q, c) \in t_{init} \cup t_{final}$, $\overleftarrow{c} = \varepsilon$, and for all $(q, a, c, q') \in T$, $\overleftarrow{c} = \varepsilon$.

The problem of the functionality for S2Cs can be decided thanks to the decidability of the functionality for non-deterministic streaming string transducers which has been shown to be in PSPACE [AD11].

We have proved that the problem of the sequentiality of functional S2Cs is in coNPTIME in [RV19] and this result is fully detailed in Chapter 5 of this thesis.

In Chapter 2, we will see that sequential S2Cs are equivalent to 1-register (hence copyless) concatenation-free DSSTs. Therefore we can reuse [AČ11] to state that the equivalence problem for sequential S2Cs is in PSPACE.

Let us discuss the equivalence problem for functional S2Cs. Functional S2Cs can also be viewed as 1-register concatenation-free non-deterministic streaming string transducers (NSST, *cf.* [AD11]). To decide the equivalence between two functional S2Cs, we can proceed using the usual technique of first testing the equality of their domains and then testing the functionality of their disjoint union. Testing the equality of two non-deterministic finite-state automata is in PSPACE. Finally, the disjoint union of two 1-register NSSTs is still a 1-register NSST[2], hence it is copyless, and its functionality can be decided in PSPACE

2. We use the same register for the two parts of the disjoint union.

[AD11]. Therefore, the equivalence problem for functional S2Cs is in PSPACE.

Again, the undecidability of the problem of the equivalence for S2Cs in general stems from the undecidability of the equivalence for FSTs [Iba77].

1.6 Summary

We have seen four different models of transducers that will be used in this thesis: finite-state transducers, two-way finite-state transducers, deterministic streaming string transducers and string-to-context transducers. In Chapter 2, we will compare the expressiveness of these models. However, we can already recapitulate the latest results on the classical problems mentioned throughout this chapter. They are summarised in Table 1.1. A recent survey can be found in [MP19].

Class of transducer	Functionality	Sequentiality	Equivalence
Sequential FST	n/a	n/a	NLOGSPACE-c [BH79]
Functional FST	n/a	NLOGSPACE-c [FMR18]	PSPACE-c [BH77]
FST	NLOGSPACE-c [FMR18]	NLOGSPACE-c [FMR18]	undecidable [Iba77]
Sequential 2FST	n/a	n/a	PSPACE-c [Gur80]
Functional 2FST	n/a	always true	PSPACE-c [Gur80]
2FST	PSPACE-c [CK87]	PSPACE-c [EH01; CK87]	undecidable
Copyless DSST	n/a	n/a	PSPACE [AČ11]
Copyful DSST	n/a	n/a	decidable [FR17]
Sequential S2C	n/a	n/a	PSPACE [AČ11]
Functional S2C	n/a	coNPTIME [RV19]	PSPACE [AD11]
S2C	PSPACE [AD11]	PSPACE [AD11; RV19]	undecidable

Table 1.1 – Summary of the results for the classical problems. "n/a" means non-applicable.

Chapter 2

Comparison of Expressiveness

In this chapter, we will study the expressiveness of the functional finite-state transducer models that we introduced in Chapter 1. Specifically, we will look at how they compare to the deterministic streaming string transducer model and its different restrictions (copyless/copyful, appending/concatenation-free).

2.1 Functional Two-way Finite-State Transducers

As defined in Section 1.3, the class of functions realised by functional 2FSTs is the class of regular functions [EH01].

[AČ10] introduced the copyless DSST model and proved that the class of functions it realises is exactly the class of regular functions, by providing constructions from sequential 2FSTs to copyless DSSTs and from copyless DSSTs to deterministic Monadic Second Order Transductions.

From these results, we draw the following proposition.

Proposition 2.1. *Let A, B be two alphabets. Let f be a function from A^* to B^*. The following assertions are equivalent:*

1. *f is a regular function,*
2. *f can be realised by a functional 2FST,*
3. *f can be realised by a copyless DSST.*

Remark. Some copyful DSSTs are not within the class of regular functions. Indeed, copyfulness gives the ability to describe transductions whose output grows non-linearly *w.r.t.* the growth of their input word. For example, Figure 2.1 depicts a copyful DSST that realises the function $u \in \{\mathsf{a}\}^* \mapsto \mathsf{a}^{2^{|u|}}$, whose output obviously grows exponentially.

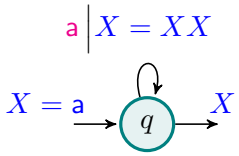


Figure 2.1 – A copyful DSST with exponential size increase.

2.2 Functional Finite-State Transducers

As defined in Section 1.2, the class of functions realised by functional FSTs is the class of rational functions [Ber13]. The following proposition extends this connection to copyful appending DSSTs, using a construction similar to [Alu+13].

Proposition 2.2. *Let A, B be two alphabets. Let f be a function from A^* to B^*. The following assertions are equivalent:*

1. *f is a rational function,*
2. *f can be realised by a functional FST,*
3. *f can be realised by a copyful appending DSST.*

The equivalence between 1 and 2 comes from the definition of rational functions. We will detail the equivalence between 2 and 3 in the next two subsections as it will be useful when considering S2Cs.

Remark. In contrast to regular functions, we need copyfulness to fully express the rational functions with appending DSSTs. We will see in Section 2.4 that, if we only use copyless appending DSSTs, we loose some form of non-determinism and obtain the class of multi-sequential rational functions.

2.2.1 From Copyful Appending DSSTs to Functional FSTs

Let $\mathcal{S} = (Q, \mathcal{X}, q_{\mathit{init}}, \nu, \delta, \mu)$ be a copyful appending DSST. We build an equivalent functional FST $\mathcal{T} = (Q', t_{\mathit{init}}, t_{\mathit{final}}, T)$.

States The states of $\mathcal{T}$ are pairs of a state and a register of $\mathcal{S}$, *i.e.* $Q' = Q \times \mathcal{X}$. Each state of $\mathcal{T}$ thus designates a register of $\mathcal{S}$, the content of which has already been outputted.

Initial Function Initial states of $\mathcal{T}$ must produce the initial valuation for their designated register.

$$\text{for all } X \in \mathcal{X} \text{ such that } \nu(X) = w, \text{ then } \xrightarrow[\mathcal{T}]{w} (q_{\mathit{init}}, X)$$

Transitions We add a transition in $\mathcal{T}$ for each assignation $Y = X \cdot w$ in $\mathcal{S}$. These transitions must produce the word that is appended to the designated register of their source state.

$$\begin{gathered}\text{for all } p, q \in Q,\ a \in A,\ w \in B^* \text{ and } X, Y \in \mathcal{X},\\ \text{such that } p \xrightarrow[\mathcal{S}]{a|\sigma} q \text{ and } \sigma(Y) = X \cdot w, \text{ then } (p, X) \xrightarrow[\mathcal{T}]{a|w} (q, Y)\end{gathered}$$

Final Function Final states of $\mathcal{T}$ correspond to final states of $\mathcal{S}$. They must also produce the word appended to their designated register, if any.

$$\text{for all } p \in Q,\ w \in B^* \text{ and } X \in \mathcal{X} \text{ such that } p \xrightarrow[\mathcal{S}]{X \cdot w}, \text{ then } (p, X) \xrightarrow[\mathcal{T}]{w}$$

The following lemma states that $\mathcal{S}$ and $\mathcal{T}$ are equivalent. As $\mathcal{S}$ is functional by definition, so is $\mathcal{T}$, and this also proves the implication from 3 to 2 of Proposition 2.2.

Lemma 2.3. $[\![\mathcal{T}]\!] = [\![\mathcal{S}]\!]$.

Proof. We first state the following construction invariant:

$$\begin{gathered}\forall u \in A^*,\ \xrightarrow[\mathcal{T}]{w_1} (q_{\mathit{init}}, X) \xrightarrow[\mathcal{T}]{u|w_2} (p, Y)\\ \text{iff } \xrightarrow[\mathcal{S}]{\nu} q_{\mathit{init}} \xrightarrow[\mathcal{S}]{u|\sigma} p \text{ with } \nu(X) = w_1 \text{ and } \sigma(Y) = X w_2\end{gathered}$$

This invariant can be proven by induction on the length of u. Both the base case and the induction step hold by construction of $\mathcal{T}$. Finally, we obtain the result by definition of the final function of $\mathcal{T}$. □

2.2.2 From Functional FSTs to Copyful Appending DSSTs

Let $\mathcal{T} = (Q, t_{init}, t_{final}, T)$ be a functional FST. We now build an equivalent copyful appending DSST $\mathcal{S} = (Q', \mathcal{X}, q_{init}, \nu, \delta, \mu)$. Intuitively, this construction extends the classical power set construction used to determinise a non-deterministic finite-state automata.

States The states of $\mathcal{S}$ are subsets of states of $\mathcal{T}$, *i.e.* $Q' = 2^Q$.

Registers We use one register per state of $\mathcal{T}$ to store the output $\mathcal{T}$ would have produced before reaching each of these states.

$$\mathcal{X} = \{X_q \mid q \in Q\}$$

Initial State The initial state of $\mathcal{S}$ is the set of initial states of $\mathcal{T}$. We define the corresponding initial valuation accordingly.

$$q_{init} = \mathsf{dom}(t_{init}) \text{ and } \nu = \{X_q \mapsto w \mid \xrightarrow[\mathcal{T}]{w} q\}$$

Transitions Given a state of $\mathcal{S}$ and a letter $a \in A$, we identify the set of transitions of $\mathcal{T}$ that are enabled, compute the new state, and update the registers accordingly. We define δ as follows:

$$\text{for all } S_1 \in Q' \text{ and } a \in A, \text{ such that } S_2 = \{q_2 \mid \exists q_1 \in S_1 \wedge q_1 \xrightarrow[\mathcal{T}]{a|w} q_2\} \neq \emptyset,$$
$$\text{then } S_1 \xrightarrow[\mathcal{S}]{a|\sigma_1 \cup \sigma_2} S_2 \text{ where}$$
$$\sigma_1 = \{X_{q_2} = X_{q_1} \cdot w \mid \exists q_1 \in S_1 \wedge q_1 \xrightarrow[\mathcal{T}]{a|w} q_2\} \text{ and } \sigma_2 = \{X_q = X_q \mid q \in Q \setminus S_2\}$$

Note that σ_1 and σ_2 are disjoint: σ_1 defines updates for the registers $\{X_q \mid q \in S_2\}$, while σ_2 defines updates for the registers $\{X_q \mid q \notin S_2\}$. Furthermore, because $\mathcal{T}$ is functional, we can arbitrarily choose any pair $(q_1, w) \in S_1 \times B^*$ such that $q_1 \xrightarrow[\mathcal{T}]{a|w} q_2$. Therefore σ_1 is well defined. Finally, as there is no transition between states $q_1 \in S_1$ and states $q \in Q \setminus S_2$, we know that the current values of the registers $\{X_q \mid q \in Q \setminus S_2\}$ won't be used in the final output. As their value is useless, we choose to simply copy them.

Final States Every state of $\mathcal{S}$ that contains a final state of $\mathcal{T}$ is final. Thus, we define μ as follows:

$$\text{for all } S \in Q' \text{ such that } S \cap \mathsf{dom}(t_{final}) \neq \emptyset,$$
$$\text{let } q \in S \text{ such that } q \xrightarrow[\mathcal{T}]{w}, \text{ then } S \xrightarrow[\mathcal{S}]{X_q \cdot w}$$

As $\mathcal{T}$ is functional, we can arbitrarily choose any pair $(q, w) \in S \times B^*$ such that $q \xrightarrow[\mathcal{T}]{w}$. Therefore μ is well defined.

The following lemma proves the implication from 2 to 3 of Proposition 2.2.

Lemma 2.4. $[\![\mathcal{S}]\!] = [\![\mathcal{T}]\!]$.

Proof. We first state the following construction invariant:

$$\forall u \in A^*, \xrightarrow[\mathcal{S}]{\nu} q_{init} \xrightarrow[\mathcal{S}]{u|\sigma} S \text{ with } \nu(X_p) = w_1 \text{ and } \sigma(X_q) = X_p w_2$$
$$\text{iff } \xrightarrow[\mathcal{T}]{w_1} p \xrightarrow[\mathcal{T}]{u|w_2} q \text{ with } p \in q_{init} \text{ and } q \in S$$

The proof can be done by induction on the length of u. Both the base case and the induction step hold by construction of $\mathcal{S}$. We also use the functionality of $\mathcal{T}$ to show that our construction is correct. Finally, we obtain the result by definition of the final states of $\mathcal{S}$. □

Observations First, note that $\mathcal{S}$ is appending by construction, as all of its register updates are of the form $Y = Xu$ for some $u \in B^*$.

Second, note that $\mathcal{S}$ may be copyful. For example, suppose that there exist some states $p, q_1, q_2 \in Q$ with $q_1 \neq q_2$, and two transitions $p \xrightarrow{a|w_1} q_1$ and $p \xrightarrow{a|w_2} q_2$ in $\mathcal{T}$ reading the same letter $a \in A$. This will result in a transition reading a from a state $S \ni p$ of $\mathcal{S}$ having parallel updates of the form $X_{q_1} = X_p \cdot w_1$ and $X_{q_2} = X_p \cdot w_2$.

2.3 Sequential Finite-State Transducers

The equivalence between the class of 1-register appending DSSTs and the class of sequential FSTs is even easier. As both models are deterministic, we only need to do syntactic rewrite of the machine's output labels, as shown in Table 2.1. This leads to the following proposition.

Proposition 2.5. *Let A, B be two alphabets. A function f from A^* to B^* can be realised by a 1-register appending DSST iff it can be realised by a sequential FST.*

Label	1-register appending DSST	sequential FST
Initial	$X = w$	w
Transition	$X = Xw$	w
Final	Xw	w

Table 2.1 – Synctactic rewrites of labels between 1-register appending DSSTs and sequential FSTs

Observations It is obvious that a 1-register appending DSST is copyless. Also, as stated before, a sequential FST is always functional.

2.4 Multi-Sequential Functional Finite-State Transducers

Proposition 2.6. *Let A, B be two alphabets. A function f from A^* to B^* can be realised by a copyless appending DSST with k registers iff it can be realised by a k-sequential functional FST.*

The equivalence is shown in the following two subsections.

2.4.1 From Copyless Appending DSSTs to Multi-Sequential Functional FSTs

Let $\mathcal{S} = (Q, \mathcal{X}, q_{init}, \nu, \delta, \mu)$ be a copyless appending DSST with $\mathcal{X} = \{X_1, \dots, X_k\}$ for some $k \in \mathbb{N}$. We build an equivalent k-sequential functional FST $\mathcal{T} = \cup_{i\in\{1,\dots,k\}}\mathcal{T}_i$ with $\mathcal{T}_i = (Q'_i, t^i_{init}, t^i_{final}, T_i)$ for $i \in \{1, \dots, k\}$. We build each $\mathcal{T}_i$ from $\mathcal{S}$ with the construction of Section 2.2.1 but keeping only (q_{init}, X_i) as its unique initial state and trimming it appropriately.

The following lemma proves the forward implication of Proposition 2.6.

Lemma 2.7. *$\mathcal{T}_i$ is sequential, for all $i \in \{1, \dots, k\}$, and $[\![\mathcal{T}]\!] = [\![\mathcal{S}]\!]$.*

Proof. As $\mathcal{S}$ is deterministic, for all $p \in Q$ and $a \in A$, there is only one transition $p \xrightarrow{a|\sigma} q$, for some $\sigma \in \mathcal{U}pd(\mathcal{X}, B)$. As $\mathcal{S}$ is copyless, for all $X \in \mathcal{X}$, there is only one register Y such that $\sigma(Y) = Xw$, for some $w \in B^*$. We obtain that for all $p \in Q$, $X \in \mathcal{X}$ and $a \in A$, if $(p, X) \xrightarrow[\mathcal{T}_i]{a|w} (q, Y)$ and $(p, X) \xrightarrow[\mathcal{T}_i]{a|w'} (q', Y')$ then $w = w'$, $q = q'$ and $Y = Y'$. Therefore $\mathcal{T}_i$ is sequential.

We now prove the equivalence between $\mathcal{T}$ and $\mathcal{S}$. Let $\mathcal{T}'$ the functional FST equivalent to $\mathcal{S}$, obtained by the construction of Section 2.2.1. This construction is such that the initial states of $\mathcal{T}'$ are $\{(q_{init}, X_i) \mid i \in \{1, \dots, k\}\}$, *i.e.* the initial states of the $\mathcal{T}_i$'s. Therefore, we obtain that $[\![\mathcal{T}]\!] = [\![\mathcal{T}']\!]$ and thus $[\![\mathcal{T}]\!] = [\![\mathcal{S}]\!]$. □

2.4.2 From Multi-Sequential Functional FSTs to Copyless Appending DSSTs

Let $\mathcal{T} = \cup_{i\in\{1,\dots,k\}}\mathcal{T}_i$ be a k-sequential functional FST where for all $i \in \{1, \dots, k\}$, $\mathcal{T}_i = (Q'_i, t^i_{init}, t^i_{final}, T_i)$.

From each of the $\mathcal{T}_i$, we build an equivalent appending DSST with 1 register, using the construction of Section 2.3. We then make the product of these k

DSSTs and we obtain an appending DSST $\mathcal{S}$ with k registers. It can easily be shown that $[\![\mathcal{S}]\!] = [\![\mathcal{T}]\!]$.

As every update of $\mathcal{S}$ is of the form $X_i = X_i w$, for some $i \in \{1, \dots, k\}$ and $w \in B^*$, we obtain that $\mathcal{S}$ is copyless, therefore proving the reverse implication of Proposition 2.6.

2.5 Functional String-to-Context Transducers

In order to better understand the expressiveness of functional S2Cs, it is useful to view them as built from two functional FSTs.

Proposition 2.8. *Let A, B be two alphabets. A function f from A^* to B^* can be realised by an S2C iff there exist two rational functions g, h from A^* to B^* such that, for all $u \in \mathsf{dom}(f)$, $f(u) = \widetilde{g(u)} \cdot h(u)$.*

Proof. Let $\mathcal{T}_f$ be an S2C realising f. We consider $\mathcal{T}_f$ to be unambiguous. If it is not, as it is functional, we can use classical automata techniques to build an unambiguous equivalent. From $\mathcal{T}_f$, we can easily build two FSTs $\mathcal{T}_g$ and $\mathcal{T}_h$ realising g and h. First, they all have the same set of states. Second, we define their transitions, and initial and final functions as follows:

$$\text{for all } p \xrightarrow[\mathcal{T}_f]{a|(u,v)}, \text{ we set } p \xrightarrow[\mathcal{T}_g]{a|\tilde{u}} \text{ and } p \xrightarrow[\mathcal{T}_h]{a|v}.$$
$$\text{for all } p \xrightarrow[\mathcal{T}_f]{a|(u,v)} q, \text{ we set } p \xrightarrow[\mathcal{T}_g]{a|\tilde{u}} q \text{ and } p \xrightarrow[\mathcal{T}_h]{a|v} q.$$
$$\text{for all } q \xrightarrow[\mathcal{T}_f]{a|(u,v)}, \text{ we set } q \xrightarrow[\mathcal{T}_g]{a|\tilde{u}} \text{ and } q \xrightarrow[\mathcal{T}_h]{a|v}.$$

As $\mathcal{T}_f$ is unambiguous, so are $\mathcal{T}_g$ and $\mathcal{T}_h$, and thus also functional. It is then easy to prove that, indeed, for all $u \in \mathsf{dom}(f)$, $f(u) = \widetilde{g(u)} \cdot h(u)$.

Conversely, from two FSTs $\mathcal{T}_g$ and $\mathcal{T}_h$ realising g and h, we can build an S2C $\mathcal{T}_f$ by applying a product construction of $\mathcal{T}_g$ and $\mathcal{T}_h$. □

And again, S2Cs are still comparable to a particular restriction of DSSTs.

Proposition 2.9. *Let A, B be two alphabets. A function f from A^* to B^* can be realised by a copyful concatenation-free DSST iff it can be realised by an S2C.*

The equivalence is shown in the following two subsections. They follow exactly the same course than Section 2.4. While appending updates correspond to productions of an FST, concatenation-free updates correspond to those of an S2C.

2.5.1 From Copyful Concatenation-Free DSSTs to Functional S2Cs

Let $\mathcal{S} = (Q, \mathcal{X}, q_{init}, \nu, \delta, \mu)$ be a copyful concatenation-free DSST. We build an equivalent functional S2C $\mathcal{T} = (Q', t_{init}, t_{final}, T)$.

States The states of $\mathcal{T}$ are pairs of a state and a register of $\mathcal{S}$, *i.e.* $Q' = Q \times \mathcal{X}$. Each state of $\mathcal{T}$ thus designates a register of $\mathcal{S}$, the content of which has already been outputted.

Initial Function Initial states of $\mathcal{T}$ must produce the initial valuation for their designated register.

$$\text{for all } X \in \mathcal{X} \text{ such that } \nu(X) = w, \text{ then } \xrightarrow[\mathcal{T}]{(\varepsilon,w)} (q_{init}, X)$$

Transitions We add a transition in $\mathcal{T}$ for each assignation $Y = v \cdot X \cdot w$ in $\mathcal{S}$. These transitions must produce the context that is added around the designated register of their source state.

$$\begin{gathered}\text{for all } p, q \in Q,\ a \in A,\ v, w \in B^* \text{ and } X, Y \in \mathcal{X},\\ \text{such that } p \xrightarrow[\mathcal{S}]{a|\sigma} q \text{ and } \sigma(Y) = v \cdot X \cdot w, \text{ then } (p, X) \xrightarrow[\mathcal{T}]{a|(v,w)} (q, Y)\end{gathered}$$

Final Function Final states of $\mathcal{T}$ correspond to final states of $\mathcal{S}$. They must also produce the word appended to their designated register, if any.

$$\text{for all } p \in Q,\ v, w \in B^* \text{ and } X \in \mathcal{X} \text{ such that } p \xrightarrow[\mathcal{S}]{v \cdot X \cdot w}, \text{ then } (p, X) \xrightarrow[\mathcal{T}]{(v,w)}$$

The following lemma states that $\mathcal{S}$ and $\mathcal{T}$ are equivalent. As $\mathcal{S}$ is functional by definition, so is $\mathcal{T}$, and this also proves the forward implication Proposition 2.9.

Lemma 2.10. $[\![\mathcal{T}]\!] = [\![\mathcal{S}]\!]$.

Proof. We first state the following construction invariant:

$$\begin{gathered}\forall u \in A^*,\ \xrightarrow[\mathcal{T}]{(\varepsilon,w_1)} (q_{init}, X) \xrightarrow[\mathcal{T}]{u|(v_2,w_2)} (p, Y)\\ \text{iff } \xrightarrow[\mathcal{S}]{\nu} q_{init} \xrightarrow[\mathcal{S}]{u|\sigma} p \text{ with } \nu(X) = w_1 \text{ and } \sigma(Y) = v_2 X w_2\end{gathered}$$

We can prove it by induction on the length of u. Both the base case and the induction step hold by construction of $\mathcal{T}$. Finally, we obtain the result by definition of the final function of $\mathcal{T}$. □

2.5.2 From Functional S2Cs to Copyful Concatenation-Free DSSTs

Let $\mathcal{T} = (Q, t_{init}, t_{final}, T)$ be a functional S2C. We now build an equivalent copyful concatenation-free DSST $\mathcal{S} = (Q', \mathcal{X}, q_{init}, \nu, \delta, \mu)$.

States The states of $\mathcal{S}$ are subsets of states of $\mathcal{T}$, *i.e.* $Q' = 2^Q$.

Registers We use one register per state of $\mathcal{T}$ to store the output $\mathcal{T}$ would have produced before reaching each of these states.

$$\mathcal{X} = \{X_q \mid q \in Q\}$$

Initial State The initial state of $\mathcal{S}$ is the set of initial states of $\mathcal{T}$. We define the corresponding initial valuation accordingly.

$$q_{init} = \mathsf{dom}(t_{init}) \text{ and } \nu = \{X_q \mapsto vw \mid \xrightarrow[\mathcal{T}]{(v,w)} q\}$$

Transitions We define δ as follows:

$$\text{for all } S_1 \in Q' \text{ and } a \in A, \text{ such that } S_2 = \{q_2 \mid \exists q_1 \in S_1 \wedge q_1 \xrightarrow[\mathcal{T}]{a|(v,w)} q_2\} \neq \emptyset,$$
$$\text{then } S_1 \xrightarrow[\mathcal{S}]{a|\sigma_1 \cup \sigma_2} S_2 \text{ where}$$
$$\sigma_1 = \{X_{q_2} = v \cdot X_{q_1} \cdot w \mid \exists q_1 \in S_1 \wedge q_1 \xrightarrow[\mathcal{T}]{a|(v,w)} q_2\} \text{ and } \sigma_2 = \{X_q = X_q \mid q \in Q \setminus S_2\}$$

We can show that $\sigma_1 \cup \sigma_2$ is well-defined, using similar arguments to the ones used in the construction from functional FSTs to copyful appending DSSTs of Section 2.2.2.

Final States Every state of $\mathcal{S}$ that contains a final state of $\mathcal{T}$ is final. Thus, we define μ as follows:

$$\text{for all } S \in Q' \text{ such that } S \cap \mathsf{dom}(t_{final}) \neq \emptyset,$$
$$\text{let } q \in S \text{ such that } q \xrightarrow[\mathcal{T}]{(v,w)}, \text{ then } S \xrightarrow[\mathcal{S}]{v \cdot X_q \cdot w}$$

As $\mathcal{T}$ is functional, we can arbitrarily choose any pair $(q, (v, w)) \in S \times B^*$ such that $q \xrightarrow[\mathcal{T}]{(v,w)}$. Therefore μ is well defined.

The following lemma proves the implication from 2 to 3 of Proposition 2.2.

Lemma 2.11. $[\![\mathcal{S}]\!] = [\![\mathcal{T}]\!]$.

Proof. We first state the following construction invariant:

$$\forall u \in A^*,\ \xrightarrow[\mathcal{S}]{\nu} q_{init} \xrightarrow[\mathcal{S}]{u|\sigma} S \text{ with } \nu(X_p) = w_1 \text{ and } \sigma(X_q) = v_2 X_p w_2$$
$$\text{iff } \xrightarrow[\mathcal{T}]{(\varepsilon,w_1)} p \xrightarrow[\mathcal{T}]{u|(v_2,w_2)} q \text{ with } p \in q_{init} \text{ and } q \in S$$

It can be proven by induction on the length of u. Both the base case and the induction step hold by construction of $\mathcal{S}$. Finally, we obtain the result by definition of the final states of $\mathcal{S}$. □

Observations First, note that $\mathcal{S}$ is concatenation-free by construction, as all of its register updates are of the form $Y = vXw$ for some $v, w \in B^*$. Second, as for the construction of Section 2.2.2, the non-determinism of $\mathcal{T}$ may yield copyful updates in $\mathcal{S}$.

2.6 Sequential String-to-Context Transducers

Similarly to Section 2.3, the equivalence between the class of 1-register concatenation-free DSSTs and the class of sequential S2Cs is even easier. As both models are deterministic, we only need to do syntactic rewrite of the machine's output labels, as shown in Table 2.2. This leads to the following proposition.

Proposition 2.12. *Let A, B be two alphabets. A function f from A^* to B^* can be realised by a 1-register concatenation-free DSST iff it can be realised by a sequential S2C.*

Label	1-register concatenation-free DSST	sequential S2C
Initial	$X = w$	(ε, w)
Transition	$X = vXw$	(v, w)
Final	vXw	(v, w)

Table 2.2 – Syntactic rewrites of labels between 1-register concatenation-free DSSTs and sequential S2Cs

Observations It is obvious that a 1-register concatenation-free DSST is copyless. Also, as stated before, a sequential S2C is always functional.

2.7 Summary

The Figure 2.2 depicts the relationships between the classes of functions realised by our different models and where our example functions are situated. We

also situate four additional example functions: $f_{mirror \cdot id}$, $f_{mirror \cdot last}$, $f_{mirror \cdot last^*}$, and $f_{id \cdot mirror}$. $f_{mirror \cdot id}$ is discussed in Example 2.1 and $f_{id \cdot mirror}$ is discussed in Example 2.2. Finally, $f_{mirror \cdot last}$ and $f_{mirror \cdot last^*}$ can be built as S2Cs in a similar way to $f_{mirror \cdot id}$ but they cannot be realised by sequential S2Cs.

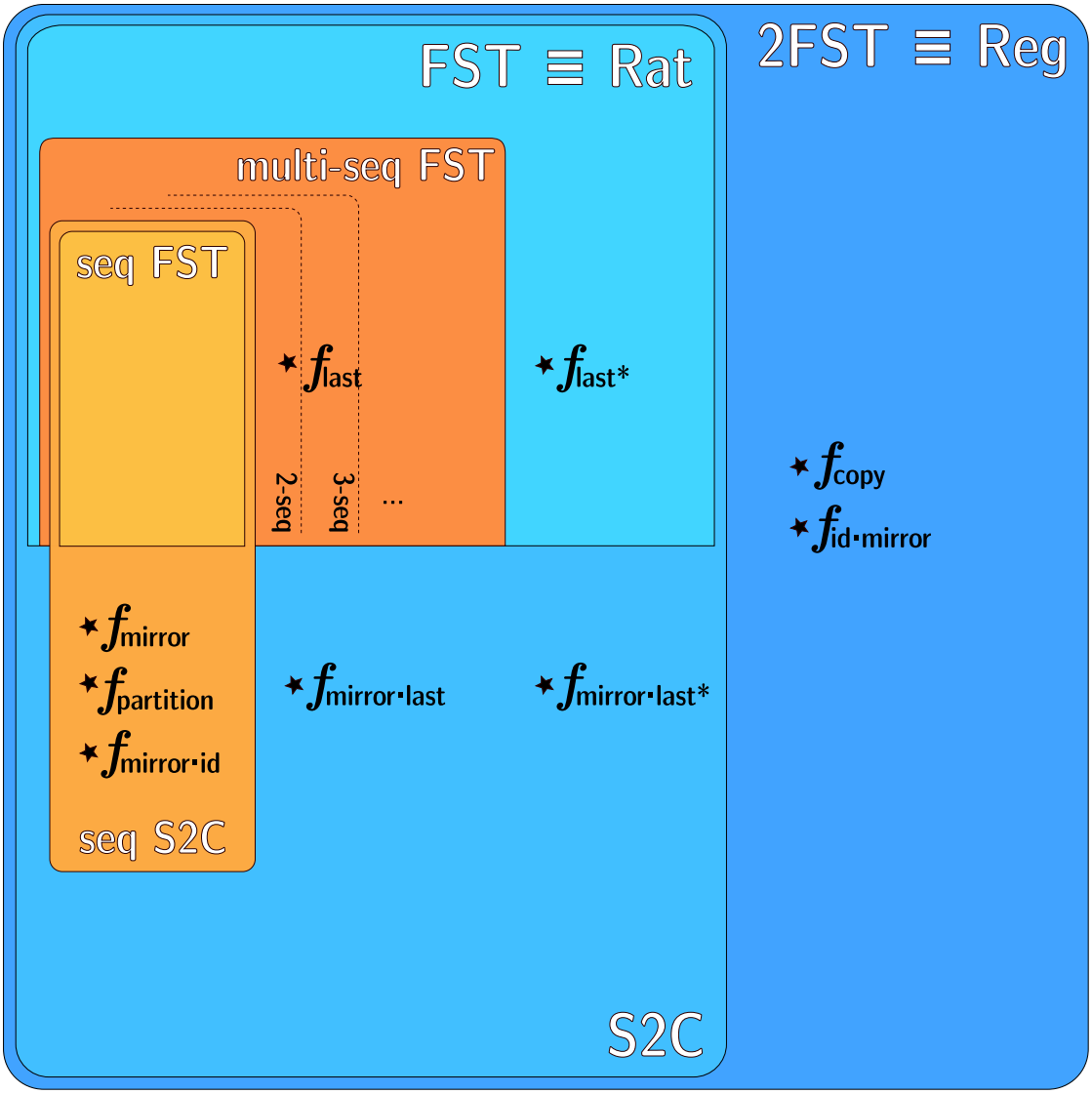
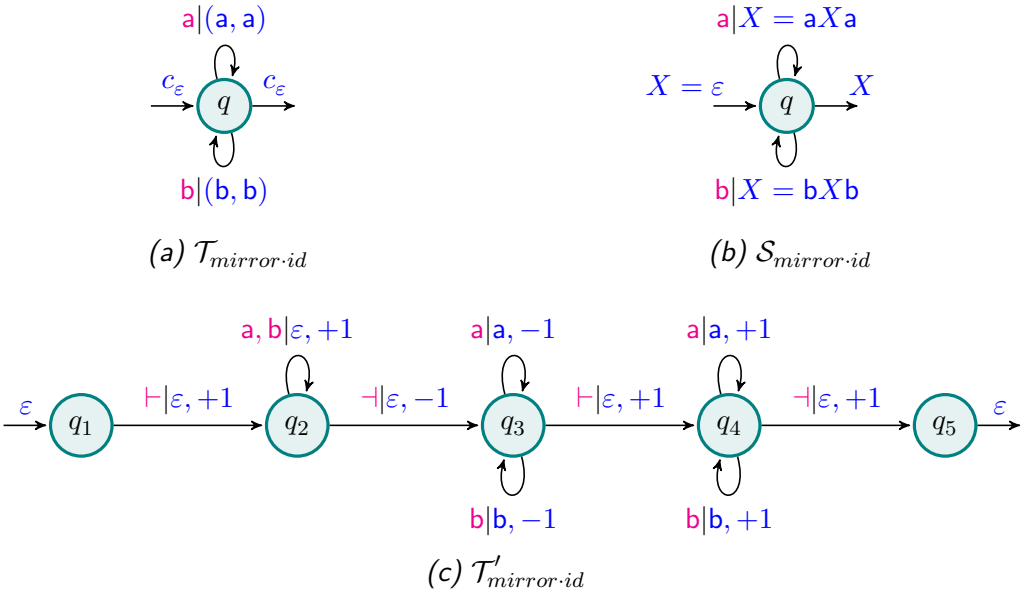


Figure 2.2 – A representation of different functional transducer classes.

Example 2.1. Three example implementations of $f_{mirror \cdot id} : u \in \{\mathsf{a}, \mathsf{b}\}^* \mapsto \tilde{u}u$ are depicted on Figure 2.3. $\mathcal{T}_{mirror \cdot id}$ is a sequential S2C that both prepends and appends the letter it reads from the input word. $\mathcal{S}_{mirror \cdot id}$ is a concatenation-free DSST with only one register that both prepends and appends the letter it reads from the input word to its register X. $\mathcal{T}'_{mirror \cdot id}$ is a 2FST. It moves forward to the end of the input word while producing no output (in state q_2), then moves

backward while copying the input word in reverse (in state q_3), and finally moves forward again back to the end of the input word while copying it a second time (in state q_4). We can prove, by using the notion of inversion defined in [Bas+18], that the function $f_{mirror \cdot id}$ cannot be realised by a one-way FST.



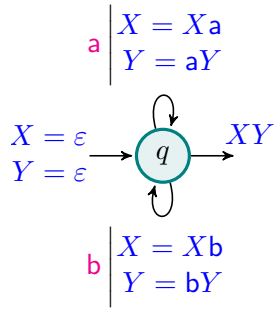
(a) $\mathcal{T}_{mirror \cdot id}$ (b) $\mathcal{S}_{mirror \cdot id}$

(c) $\mathcal{T}'_{mirror \cdot id}$

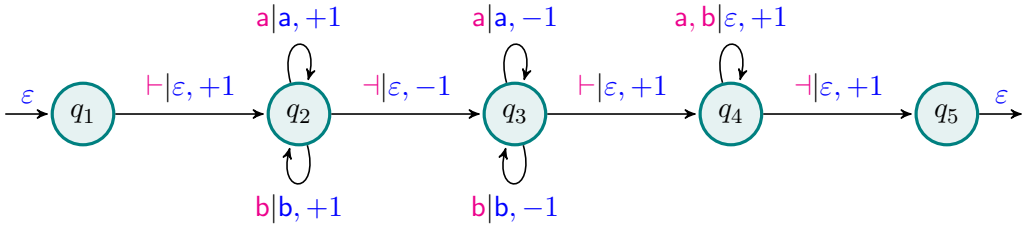
Figure 2.3 – Three example implementations of $f_{mirror \cdot id}$.

Example 2.2. Two example implementations of $f_{id \cdot mirror} : u \in \{\mathsf{a}, \mathsf{b}\}^* \mapsto u\tilde{u}$ are depicted on Figure 2.4. $\mathcal{S}_{id \cdot mirror}$ is a DSST with two registers X and Y. It simultaneously appends the read letter to X and prepends it to Y. The final output is obtained by concatenating X and Y. $\mathcal{T}_{id \cdot mirror}$ is a 2FST. It moves forward while copying the input word a first time (in state q_2), then moves backward while copying the input word in reverse (in state q_3), and finally moves forward again back to the end of the input word while producing no output (in state q_4). The function $f_{id \cdot mirror}$ cannot be realised by a DSST with one register nor by an S2C. This can easily be understood by considering the characterisation of S2Cs from Proposition 2.8.

The next three chapters will discuss how to characterise functions (and the transducers realising those) in the three classes depicted in orange (sequential FST, multi-sequential FST and sequential S2C) among the functions in the two classes depicted in light blue (FST and S2C). Chapter 3 will recall results, due to [Cho77], that allow to characterise, amongst the class of functional FSTs, the ones that admit an equivalent sequential FST. Chapter 4 will present original results, published in [Dav+17], that allow, given $k \in \mathbb{N}$, to characterise, amongst the class of functional FSTs, the ones that admit an equivalent k-sequential FST. Finally, Chapter 5 will present original results, published in [RV19], that allow



(a) $\mathcal{S}_{id\cdot mirror}$



(b) $\mathcal{T}_{id\cdot mirror}$

Figure 2.4 – Two example implementations of $f_{id\cdot mirror}$.

to characterise, amongst the class of functional S2Cs, the ones that admit an equivalent sequential S2C.

Chapter 3

Sequentiality of Finite-State Transducers

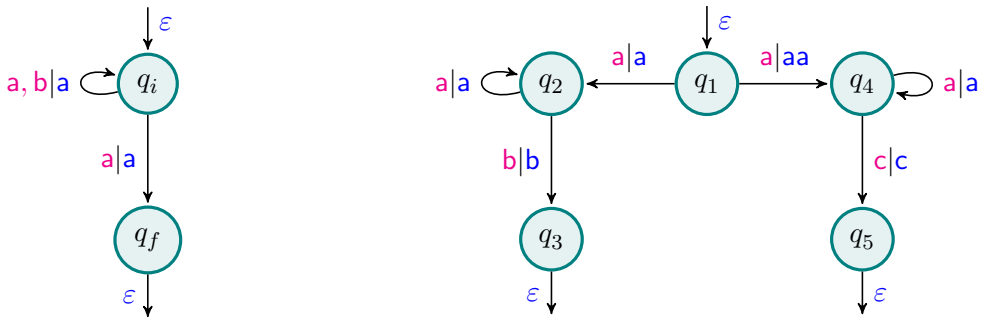
In this chapter, we recall the work of [Cho77] to characterise, amongst functional finite-state transducers, the ones that admit an equivalent sequential finite-state transducer. It will provide a basis for the understanding of the extensions that we build in Chapters 4 and 5. For another full account of those results, see [BC02].

You may recall from Chapter 1 that a sequential finite-state transducer $\mathcal{T} = (Q, t_{init}, t_{final}, T)$ has the following syntactic restriction: $\mathsf{dom}(t_{init})$ is a singleton and for every transitions $(p, a, w, q), (p, a, w', q') \in T$, we have $q = q'$ and $w = w'$. We call the class of rational functions that can be realised by a sequential finite-state transducer is called the class of sequential functions.

The following example shows two classical traits of non-determinism in FSTs.

Example 3.1. Two examples of functional, but non-sequential, finite-state transducers are depicted on Figure 3.1. $\mathcal{T}_{ending}$, on Figure 3.1a, computes the function $f_{ending} : u \in \{\mathsf{a}, \mathsf{b}\}^*\mathsf{a} \mapsto \mathsf{a}^{|u|}$. $\mathcal{T}_{ending}$ non-deterministically guesses, while in q_i, whether the current a letter is the last letter of the input word, and hence whether it should move to the final state q_f. $\mathcal{T}_{synchro}$, on Figure 3.1b, computes the function $f_{synchro}$ which maps an input word u to itself if $u \in \mathsf{a}^+\mathsf{b}$, and to $\mathsf{a} \cdot u$

if $u \in \mathsf{a}^+\mathsf{c}$. $\mathcal{T}_{synchro}$ non-deterministically guesses, while in q_1, whether the input word ends with a b or a c, and goes accordingly to the left or right branch (q_2 or q_4). It produces one more a while going to the right branch.



(a) $\mathcal{T}_{ending}$ *(b)* $\mathcal{T}_{synchro}$

Figure 3.1 – Two example FSTs: $\mathcal{T}_{ending}$ and $\mathcal{T}_{synchro}$.

Note that there was initially two different classes called sub-sequential transducers, introduced by Ginsburg et Rose, and sequential transducers, introduced by Schützenberger. In fact, the work of [Cho77] characterises the sub-sequential ones. In this thesis, we chose to use the term "sequential" to speak about the more general sub-sequential class, following the consensus in the recent literature.

In Section 3.1, we first present the properties used to characterise functions that are realisable by a sequential finite-state transducer. Then we present a pattern property of finite-state transducers that can also be realised by an equivalent sequential finite-state transducer. Finally, we present the main result of [Cho77], the so-called sequentialisation theorem, that links all these properties to the existence of an equivalent sequential finite-state transducer.

In Section 3.2, we describe and prove the construction of an equivalent sequential transducer. Lastly, in Section 3.3 we discuss the decision of the sequentiality problem for finite-state transducers.

3.1 Characterisation of Sequential Functions

We here introduce the different properties that characterise the functions that can be realised by a sequential transducer and the functional transducers that admit an equivalent sequential one.

3.1.1 Bounded Variation Property

The bounded variation property was first introduced in [Cho77], and deals with word to word functions. Given a function f, it states that if two words are

close, *w.r.t.* to their prefix distance, then their images by f are also close.

Definition 3.1 (Bounded variation property)**.** Let A, B be two alphabets. A function f from A^* to B^* satisfies the *bounded variation property* if for all $m \in \mathbb{N}$, there exists $M \in \mathbb{N}$ such that for all $u, v \in \mathsf{dom}(f)$, if $\mathsf{dist}_p(u, v) \leqslant m$ then $\mathsf{dist}_p(f(u), f(v)) \leqslant M$.

3.1.2 Lipschitz Property

The application of the Lipschitz property to rational functions was hinted in [Ber13, Example 2.8] as part of a presentation of the results of [Cho77]. Given a function f, it states that the prefix distance between the images by f of two words is proportional to the prefix distance between these two input words.

Definition 3.2 (Lipschitz property)**.** Let A, B be two alphabets. A function f from A^* to B^* satisfies the *Lipschitz property* if there exists $K \in \mathbb{N}$ such that for all $u, v \in \mathsf{dom}(f)$, $\mathsf{dist}_p(f(u), f(v)) \leqslant K.\mathsf{dist}_p(u, v)$.

Example 3.2. The function f_{ending} defined in Example 3.1 obviously satisfies the Lipschitz property with coefficient 1. Indeed, let $u, v \in \{\mathsf{a}, \mathsf{b}\}^*\mathsf{a}$. We have $f(u) = \mathsf{a}^{|u|}$ and $f(v) = \mathsf{a}^{|v|}$. Then, $\mathsf{dist}_p(f(u), f(v)) = ||u| - |v|| \leqslant \mathsf{dist}_p(u, v)$.

Similarly, one can prove that the function $f_{synchro}$ defined in Example 3.1 also satisfies the Lipschitz property.

Example 3.3. The function f_{last} defined in Example 1.3 does not satisfy the Lipschitz property. Indeed, let $K \in \mathbb{N}$ and take $u = \mathsf{a}^K\mathsf{a}$, $v = \mathsf{a}^K\mathsf{b}$. We have $f(u) = \mathsf{a}^{K+1}$ and $f(v) = \mathsf{b}^{K+1}$. Then, $\mathsf{dist}_p(f(u), f(v)) = 2K + 2 > K.\mathsf{dist}_p(u, v) = 2K$.

In the next lemma, we state that the Lipschitz property implies the bounded variation property. We will later prove that they actually are equivalent.

Lemma 3.1. *Let A, B be two alphabets. If a function f from A^* to B^* satisfies the Lipschitz property then it satisfies the bounded variation property.*

Proof. Let f that satisfies the Lipschitz property and let $K \in \mathbb{N}$ such that for all $u, v \in \mathsf{dom}(f)$, $\mathsf{dist}_p(f(u), f(v)) \leqslant K.\mathsf{dist}_p(u, v)$. We prove that f satisfies the bounded variation property. Let $m \in \mathbb{N}$ and define $M = Km$. If $u, v \in \mathsf{dom}(f)$ and $\mathsf{dist}_p(u, v) \leqslant m$ then we have $\mathsf{dist}_p(f(u), f(v)) \leqslant K.\mathsf{dist}_p(u, v) \leqslant M$. □

3.1.3 Twinning Property

We now introduce the twinning property, originally formulated by [Cho77], which defines a structural property of transducers.

Definition 3.3 (Twinning property – Choffrut's version ($\mathsf{TP}_{choffrut}$)). Two states q_1, q_2 of an FST are said to be *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, p_2 are initial states, we have either $y_1 = y_2 = \varepsilon$, or there exists a word z such that either $w_1x_1 = w_2x_2z$ and $zy_1 = y_2z$, or $w_2x_2 = w_1x_1z$ and $zy_2 = y_1z$. An FST satisfies the *twinning property* if any two of its states are twinned.

Remark. As explained in Chapter 1, we consider our FSTs to be trimmed. Therefore, in the previous definition, both q_1 and q_2 are co-accessible, *i.e.* there exist runs from q_1 and q_2 to some final states.

The following two lemmas state some consequences of the $\mathsf{TP}_{choffrut}$.

Lemma 3.2. *Let four words x_1,x_2,x'_1,x'_2 such that either $x_2 = x'_2 = \varepsilon$ or there exists a word z such that either $x_1 = x'_1z$ and $zx_2 = x'_2z$, or $x_1z = x'_1$ and $x_2z = zx'_2$, then for all words x_3, x'_3, $\mathsf{dist}_p(x_1x_2x_3, x'_1x'_2x'_3) = \mathsf{dist}_p(x_1x_3, x'_1x'_3)$.*

Proof. If $x_2 = x'_2 = \varepsilon$ then the result is trivial. We now consider that there exists a word z such that $x_1z = x'_1$ and $x_2z = zx'_2$. The other case is obtained by symmetry. Then we have $x'_1x'_2x'_3 = x_1x_2zx'_3$. Therefore $\mathsf{dist}_p(x_1x_2x_3, x'_1x'_2x'_3) = \mathsf{dist}_p(x_3, zx'_3) = \mathsf{dist}_p(x_1x_3, x'_1x'_3)$. □

Lemma 3.3. *Let $\mathcal{T} = (Q, t_{init}, t_{final}, T)$ be an FST that satisfies the $\mathsf{TP}_{choffrut}$. For any two runs $\xrightarrow{w} i \xrightarrow{u|x} q$ and $\xrightarrow{w'} i' \xrightarrow{u|x'} q'$, with i and i' initial, we have $\mathsf{dist}_p(wx, w'x') \leqslant 2M_{\mathcal{T}}(|Q|^2 + 1)$.*

Proof. We proceed by strong induction on the length of u.

If $|u| \leqslant |Q|^2$, then $\mathsf{dist}_p(wx, w'x') \leqslant |wx| + |w'x'| \leqslant 2M_{\mathcal{T}}(|Q|^2 + 1)$.

Otherwise, we can exhibit a synchronized loop in both runs

$$\xrightarrow{w} i \xrightarrow{u_1|x_1} p \xrightarrow{u_2|x_2} p \xrightarrow{u_3|x_3} q \quad \text{and} \quad \xrightarrow{w'} i' \xrightarrow{u_1|x'_1} p' \xrightarrow{u_2|x'_2} p' \xrightarrow{u_3|x'_3} q'$$

such that $u_1u_2u_3 = u$, $x_1x_2x_3 = x$, $x'_1x'_2x'_3 = x'$, and $|u_2| > 0$. By Lemma 3.2, we have $\mathsf{dist}_p(wx_1x_2x_3, w'x'_1x'_2x'_3) = \mathsf{dist}_p(wx_1x_3, w'x'_1x'_3)$. As $|u_1u_3| < |u_1u_2u_3|$, we can apply the induction hypothesis on the runs

$$\xrightarrow{w} i \xrightarrow{u_1|x_1} p \xrightarrow{u_3|x_3} q \quad \text{and} \quad \xrightarrow{w'} i' \xrightarrow{u_1|x'_1} p' \xrightarrow{u_3|x'_3} q'$$

and we obtain $\mathsf{dist}_p(wx_1x_3, w'x'_1x'_3) \leqslant 2M_{\mathcal{T}}(|Q|^2 + 1)$. □

We now introduce a slightly more abstract twinning property. It is based on the prefix distance and hides the combinatorial nature of the $\mathsf{TP}_{choffrut}$. We will use this presentation in our further developments.

Definition 3.4 (Twinning property – distance version (TP_{dist}))**.** Two states q_1, q_2 of an FST are said to be *L-twinned*, for some $L \in \mathbb{N}$, 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, p_2 are initial states, we have for all $j \geqslant 0$, $\mathsf{dist}_p(w_1x_1y_1^j, w_2x_2y_2^j) \leqslant L$. An FST satisfies the *twinning property* if there exists $L \in \mathbb{N}$ such that any two of its states are L-twinned.

The following lemma states that both definitions are equivalent.

Lemma 3.4. *An FST satisfies the* TP_{dist} *if and only if it satisfies the* $\mathsf{TP}_{choffrut}$*.*

Proof. Let $\mathcal{T}$ an FST that satisfies the TP_{dist}. We show that $\mathcal{T}$ satisfies the $\mathsf{TP}_{choffrut}$. Let $L \in \mathbb{N}$ and q_1, q_2 two L-twinned states of $\mathcal{T}$. For any two runs $\xrightarrow{x_1} p_1 \xrightarrow{u|x_2} q_1 \xrightarrow{v|x_3} q_1$ and $\xrightarrow{y_1} p_2 \xrightarrow{u|y_2} q_2 \xrightarrow{v|y_3} q_2$, we have for all $j \geqslant 0$, $\mathsf{dist}_p(x_1x_2x_3^j, y_1y_2y_3^j) \leqslant L$. Observe that $x_1x_2x_3^j$ and $y_1y_2y_3^j$ must grow at the same pace, and thus $|x_3| = |y_3|$. Consider that $x_3 \neq \varepsilon$ and $y_3 \neq \varepsilon$. If $|x_1x_2| = |y_1y_2|$ then we let $z = \varepsilon$ and we trivially obtain the result. Otherwise, we can grow j enough to find a sufficiently great factor between x_3^j and y_3^j, and therefore, by Lemma 1.1, the primitive roots of x_3 and y_3 are conjugates. Let t_1, t_2 such that $x_3 = (t_1t_2)^\alpha$ and $y_3 = (t_2t_1)^\alpha$ for some $\alpha \geqslant 1$. Finally, if $|x_1x_2| > |y_1y_2|$, we let $z = (t_2t_1)^\beta t_2$ for some $\beta \geqslant 0$ such that $x_1x_2 = y_1y_2z$, and we have $zx_3 = y_3z$. Otherwise, we let $z = (t_1t_2)^\beta t_1$ for some $\beta \geqslant 0$ such that $x_1x_2z = y_1y_2$, and we have $x_3z = zy_3$.

The other direction is obtained by Lemma 3.3, taking $L = 2M_{\mathcal{T}}(|Q|^2 + 1)$. □

From now on, as the $\mathsf{TP}_{choffrut}$ and TP_{dist} are equivalent, we will simply refer to the TP_{dist} as the "twinning property" (TP).

Example 3.4. The finite-state transducer T_{ending}, given in Figure 3.1a, that computes the function f_{ending}, obviously satisfies the twinning property. Indeed, q_i is 0-twinned with itself, as we cannot find two different runs with loops around q_i. For the same reason, so is q_f. Finally, q_i and q_f are 1-twinned, because the only synchronised runs reaching q_i and q_f have a non-productive loop.

Similarly, we can prove that the finite-state transducer $T_{synchro}$, given in Figure 3.1b, that computes the function $f_{synchro}$, also satisfies the twinning property. Indeed, the only interesting state pair is (q_2, q_4) and we can verify that they are twinned: The runs reaching q_2 and q_4 only produce a's and the loops around q_2 and q_4 have the same productions.

Example 3.5. The finite-state transducer $\mathcal{T}_{last}$, given in Figure 1.2a, that computes the function f_{last}, does not satisfy the twinning property. Indeed, in search of a contradiction, assume that $\mathcal{T}_{last}$ does satisfy the twinning property and let $L \in \mathbb{N}$ such that any two states of $\mathcal{T}_{last}$ are L-twinned. Now, consider two loops around q_a and q_b: $p_1 = q_1 = q_a$, $p_2 = q_2 = q_b$, $u = \varepsilon$ and $v = \mathsf{a}$. Then we have $w_1 = w_2 = x_1 = x_2 = \varepsilon$, $y_1 = \mathsf{a}$ and $y_2 = \mathsf{b}$. Thus $\mathsf{dist}_p(w_1x_1y_1^L, w_2x_2y_2^L) = 2L > L$ and we have a contradiction.

3.1.4 Sequentialisation Theorem

The main result of [Cho77] is the following theorem, which characterises, amongst functional finite-state transducers, the ones that admit an equivalent sequential finite-state transducer.

Theorem 3.5. *Let A, B be two alphabets. Let $\mathcal{T}$ be a functional FST from A^* to B^*. The following assertions are equivalent:*

1. *$[\![\mathcal{T}]\!]$ satisfies the Lipschitz property,*
2. *$[\![\mathcal{T}]\!]$ satisfies the bounded variation property,*
3. *$\mathcal{T}$ satisfies the twinning property,*
4. *$[\![\mathcal{T}]\!]$ can be realised by a sequential FST.*

Proof. The implication from 1 to 2 was proved in Lemma 3.1. The implications from 4 to 1 and from 2 to 3 are proved in Propositions 3.6 and 3.7. The implication from 3 to 4 involves the construction of an equivalent sequential FST which is detailed and proved in Section 3.2. Figure 3.2 depicts the proof diagram. □

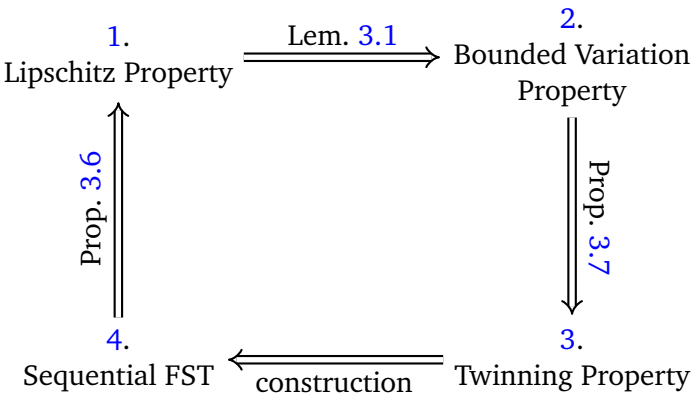


Figure 3.2 – Proof diagram of the sequentialisation theorem.

Proposition 3.6. *Let A, B be two alphabets. Let $\mathcal{T}$ be a sequential FST realising the function f from A^* to B^*. Then f satisfies the Lipschitz property.*

Proof. We will prove that f satisfies the Lipschitz property 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 $u = w.u'$ and $v = w.v'$, with $0 \leqslant |u'|$ and $0 \leqslant |v'|$. As $\mathcal{T}$ is sequential, we have two runs in $\mathcal{T}$

$$\xrightarrow{x_1} p \xrightarrow{w|x_2} q \xrightarrow{u'|y_1} r \xrightarrow{y_2} \quad \text{and} \quad \xrightarrow{x_1} p \xrightarrow{w|x_2} q \xrightarrow{v'|z_1} s \xrightarrow{z_2}$$

such that $[\![\mathcal{T}]\!](u) = x_1x_2y_1y_2$ and $[\![\mathcal{T}]\!](v) = x_1x_2z_1z_2$. We also have $|y_1| \leqslant M_{\mathcal{T}}\,|u'|$, $|z_1| \leqslant M_{\mathcal{T}}\,|v'|$, $|y_2| \leqslant M_{\mathcal{T}}$, and $|z_2| \leqslant M_{\mathcal{T}}$. Finally, as $u \neq v$, we have $\mathsf{dist}_p(u,v) = |u'| + |v'| \geqslant 1$ and we obtain:

$$\begin{aligned}\mathsf{dist}_p(f(u), f(v)) &\leqslant |y_1y_2| + |z_1z_2| \\ &\leqslant M_{\mathcal{T}}\,(2 + |u'| + |v'|) \\ &\leqslant 3M_{\mathcal{T}}\,(|u'| + |v'|) \\ &\leqslant 3M_{\mathcal{T}}\,\mathsf{dist}_p(u,v)\end{aligned}$$ □

Proposition 3.7. *Let A, B be two alphabets. Let $\mathcal{T}$ be a functional FST realising the function f from A^* to B^*. If f satisfies the bounded variation property, then $\mathcal{T}$ satisfies the twinning property.*

Proof. We denote by n the number of states of $\mathcal{T}$. Suppose that f satisfies the bounded variation property, and let $N \in \mathbb{N}$ such that for all $u, v \in \mathsf{dom}(f)$, if $\mathsf{dist}_p(u,v) \leqslant 2n$ then $\mathsf{dist}_p(f(u), f(v)) \leqslant N$.

We consider an instance of the twinning property in $\mathcal{T}$:

$$\xrightarrow{x_1} p_1 \xrightarrow{u|x_2} q_1 \xrightarrow{v|x_3} q_1 \quad \text{and} \quad \xrightarrow{y_1} p_2 \xrightarrow{u|y_2} q_2 \xrightarrow{v|y_3} q_2$$

As $\mathcal{T}$ is trimmed, there exist runs

$$q_1 \xrightarrow{w_1|x_4} r_1 \xrightarrow{x_5} \quad \text{and} \quad q_2 \xrightarrow{w_2|y_4} r_2 \xrightarrow{y_5}$$

with $|w_1| \leqslant n$ and $|w_2| \leqslant n$. We consider the input words $\alpha_j = uv^jw_1$ and $\beta_j = uv^jw_2$, for all $j \geqslant 0$. We have, for all $j \geqslant 0$, $\mathsf{dist}_p(\alpha_j, \beta_j) \leqslant |w_1| + |w_2| \leqslant 2n$. Therefore, for all $j \geqslant 0$, $\mathsf{dist}_p(f(\alpha_j), f(\beta_j)) \leqslant N$.

By using the triangle inequality twice, we obtain that, for all $j \geqslant 0$:

$$\begin{aligned}\mathsf{dist}_p(x_1x_2x_3^j, y_1y_2y_3^j) &\leqslant \mathsf{dist}_p(x_1x_2x_3^j, x_1x_2x_3^jx_4x_5) \\ &\quad + \mathsf{dist}_p(x_1x_2x_3^jx_4x_5, y_1y_2y_3^jy_4y_5) \\ &\quad + \mathsf{dist}_p(y_1y_2y_3^jy_4y_5, y_1y_2y_3^j) \\ &\leqslant \mathsf{dist}_p(x_1x_2x_3^jx_4x_5, y_1y_2y_3^jy_4y_5) + |x_4x_5| + |y_4y_5| \\ &\leqslant \mathsf{dist}_p(f(\alpha_j), f(\beta_j)) + 2(n+1)M_{\mathcal{T}} \\ &\leqslant N + 2(n+1)M_{\mathcal{T}}\end{aligned}$$ □

3.2 Construction of a Sequential Equivalent

In this section, we consider a functional FST $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$. We build an equivalent sequential FST $\mathcal{D} = (Q', t'_{\mathit{init}}, t'_{\mathit{final}}, T')$ which may have infinitely many states. We will prove that $\mathcal{D}$ is finite if $\mathcal{T}$ satisfies the twinning property.

$\mathcal{D}$ operates in a similar way to the power set construction of non-deterministic automata. For a word $u \in \mathsf{dom}(\mathcal{T})$, it computes the states of $\mathcal{T}$ that are reachable by reading u and produces the longest common prefix of the outputs of all the corresponding runs in $\mathcal{T}$. Therefore, a state of $\mathcal{D}$ stores the reachable states of $\mathcal{T}$ and, for each of these states, the remaining output word to produce.

States The states of $\mathcal{D}$ are sets of pairs of a state of $\mathcal{T}$ and a word over B, *i.e.* $Q' = 2^{Q \times B^*}$. A priori, this set is infinite.

Initial Function The initial function of $\mathcal{D}$ defines a unique initial state. It is associated to the longest common prefix of the outputs for the initial states of $\mathcal{T}$.

$$\text{let } v = \mathsf{lcp}\{w \mid (q, w) \in t_{\mathit{init}}\} \text{ and } I = \{(q, w) \mid (q, vw) \in t_{\mathit{init}}\}$$
$$\text{then } \xrightarrow[\mathcal{D}]{v} I$$

Transitions Given a state of $\mathcal{D}$ and a letter $a \in A$, we identify the set of transitions of $\mathcal{T}$ that are enabled and compute the new state. We take care of consuming the longest common prefix of their outputs and store the remaining outputs.

$$\text{for all } S_1 \in Q' \text{ and } a \in A, \text{ such that } S_1' = \{(q_2, wx) \mid (q_1, w) \in S_1 \wedge q_1 \xrightarrow[\mathcal{T}]{a|x} q_2\} \neq \emptyset,$$
$$\text{let } v = \mathsf{lcp}\{w \mid (q, w) \in S_1'\} \text{ and } S_2 = \{(q, w) \mid (q, vw) \in S_1'\}$$
$$\text{then } S_1 \xrightarrow[\mathcal{D}]{a|v} S_2$$

Final Function Every state of $\mathcal{D}$ that contains a final state of $\mathcal{T}$ is final. We take care of producing all the remaining output.

$$\text{for all } S \in Q' \text{ such that there exists } (q, w) \in S \text{ and } q \in \mathsf{dom}(t_{\mathit{final}}),$$
$$\text{let } (q, w) \in S \text{ such that } q \xrightarrow[\mathcal{T}]{x}, \text{ then } S \xrightarrow[\mathcal{D}]{wx}$$

Note that we need to choose some (q, w) in S. However, as we will see, the functionality of $\mathcal{T}$ ensures that this definition is independent of this choice.

Observe that, by definition of the initial function and the transitions, we have:

$$\text{for all } S \in Q', \mathsf{lcp}\{w \mid (q, w) \in S\} = \varepsilon \tag{P1}$$

Also, we can prove by induction that the following construction invariant holds:

$$\text{if } \xrightarrow[\mathcal{D}]{w} I \xrightarrow[\mathcal{D}]{u|x} P \text{ then } P = \{(p, y) \mid \xrightarrow[\mathcal{T}]{w'} i \xrightarrow[\mathcal{T}]{u|x'} p \wedge wxy = w'x'\} \tag{P2}$$

Therefore, if both (q, w) and (q', w') are in a state $P \in Q'$ and both $q \xrightarrow[\mathcal{T}]{x}$ and $q' \xrightarrow[\mathcal{T}]{x'}$, then we have, by functionality of $\mathcal{T}$, that $wx = w'x'$. This implies that the final function of $\mathcal{D}$ is well-defined.

Lemma 3.8. *$\mathcal{D}$ is sequential and equivalent to $\mathcal{T}$.*

Proof. By construction, $\mathcal{D}$ is sequential. Furthermore, by (P2) and the definition of the final function of $\mathcal{D}$, we have $[\![\mathcal{D}]\!] = [\![\mathcal{T}]\!]$. □

The following theorem proves the implication from 3 to 4 of Theorem 3.5.

Theorem 3.9. *If $\mathcal{T}$ satisfies the twinning property, then $\mathcal{D}$ is a finite sequential finite-state transducer equivalent to $\mathcal{T}$.*

Proof. We first prove that the words stored in the states of $\mathcal{D}$ are bounded. Consider a run $\xrightarrow{w} I \xrightarrow{u|x} P$ in $\mathcal{D}$ and a pair $(q_1, y_1) \in P$. By (P1), there exists a pair $(q_2, y_2) \in P$ such that $\mathsf{lcp}(y_1, y_2) = \varepsilon$. By (P2), there exist runs $\xrightarrow{w_1} i_1 \xrightarrow{u|x_1} q_1$ and $\xrightarrow{w_2} i_1 \xrightarrow{u|x_2} q_1$ in $\mathcal{T}$ such that $wxy_1 = w_1x_1$ and $wxy_2 = w_2x_2$. As $\mathcal{T}$ satisfies the twinning property, and by Lemma 3.3, we have $\mathsf{dist}_p(wxy_1, wxy_2) \leqslant 2M_{\mathcal{T}}(|Q|^2 + 1)$. Therefore, $|y_1| \leqslant 2M_{\mathcal{T}}(|Q|^2 + 1)$.

This in turn means that $\mathcal{D}$ is finite. By Lemma 3.8, we obtain the result. □

Example 3.6. Figure 3.1 depicts the sequential FSTs $\mathcal{D}_{ending}$ and $\mathcal{D}_{synchro}$ built with the construction from $\mathcal{T}_{ending}$ and $\mathcal{T}_{synchro}$. Observe how $\mathcal{D}_{ending}$ remembers whether the last read letter is an a or a b, and how $\mathcal{D}_{synchro}$ stores the additional a output letter in case the last letter of the input word would be a c.

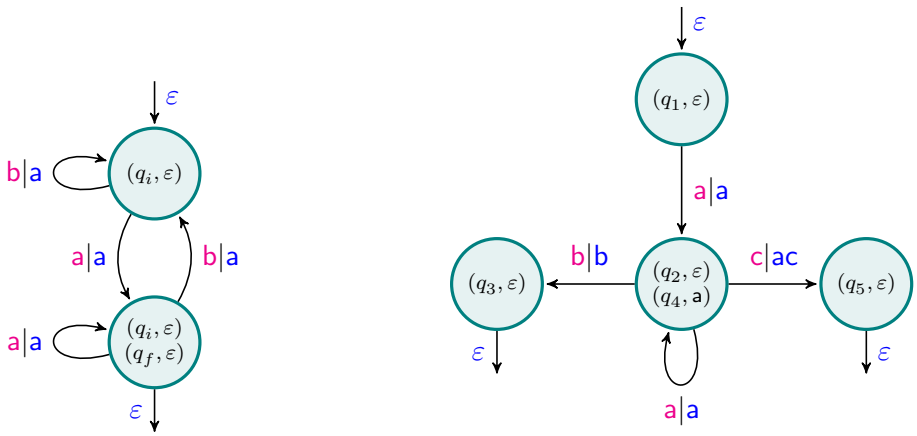


Figure 3.3 – Two built sequential FSTs $\mathcal{D}_{ending}$ and $\mathcal{D}_{synchro}$.

3.3 Deciding Sequentiality

In this section, we discuss the decision of the following problem:

Problem 3.1 (Sequentiality)**.** Given a functional finite-state transducer $\mathcal{T}$, does $\mathcal{T}$ admit an equivalent sequential finite-state transducer?

We will first give a historical perspective on how this problem has been solved. Then we will highlight a key point of the decision procedure, that we will reuse in the further developments of this thesis.

Thanks to Theorem 3.5, deciding sequentiality is equivalent to deciding the twinning property:

Problem 3.2 (TP)**.** Given a functional finite-state transducer $\mathcal{T}$, does $\mathcal{T}$ satisfy the twinning property?

As formulated in Definition 3.3, for $\mathcal{T}$ to satisfy the twinning property, any two of its states have to be twinned. Also, two states q_1, q_2 are twinned if the outputs of any two synchronised runs looping around q_1 and q_2 satisfy a particular combinatorial property (*cf.* Definition 3.3). Given a pair of states (q_1, q_2), [Cho77] proved that it suffices to check this combinatorial property for any two synchronised runs reading an input word of length at most $2n^2$, where n is the number of states $\mathcal{T}$. Therefore, the twinning property is decidable.

[WK94] proved it is is PTIME by using graph techniques. [Béa+00; BC02] then did two other presentations of this result, one by verifying a property directly on the accessible part of the square of the transducer and the other based on the decidability in polynomial time of the functionality over infinite words, also resulting in PTIME algorithms. More recently, [FMR18] devised a logic to express structural properties of automata such as the twinning property, leading to a decision procedure in NLOGSPACE.

We now propose another presentation of the decision procedure, using some techniques adapted from the work of [WK94], and highlighting the fact it is in NLOGSPACE. We will use these ideas later in this thesis.

We say that there is a *mismatch* between two words if there exists a position at which they differ. Let L be a positive integer. We say that two runs ρ_1 and ρ_2 on the same input word u are *L-close* if, for every prefix u' of u, the restrictions ρ_1' and ρ_2' of the two runs on the input u' are such that $\mathsf{dist}_p(\mathsf{out}(\rho_1'), \mathsf{out}(\rho_2')) \leqslant L$.

Lemma 3.10. *Let $\mathcal{T}$ be a finite-state transducer. $\mathcal{T}$ violates the twinning property iff there are 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, p_2 are initial states, such that*

a) either $|y_1| \neq |y_2|$

b) or $|y_1| = |y_2| \neq 0$, and there is a mismatch between the words w_1x_1 and w_2x_2, and the runs $\xrightarrow{w_1} p_1 \xrightarrow{u|x_1} q_1$ and $\xrightarrow{w_2} p_2 \xrightarrow{u|x_2} q_2$ are $2M_{\mathcal{T}}(|Q|^2+1)$-close.

Proof. The reverse implication is trivial, so we focus on the direct one. We consider a counter-example to the twinning property and aim at deriving a counter example satisfying the above properties.

Let $L \in \mathbb{N}$, and q_1, q_2 be two states such that there are 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, p_2 are initial states, such that there exists $j \in \mathbb{N}$ such that $\mathsf{dist}_p(w_1x_1y_1^j, w_2x_2y_2^j) > L$.

Either, the distance is due to the length of the output words, *i.e.* $|y_1| \neq |y_2|$, then we are in case a). Otherwise, there are two cases. If there is a mismatch between the words w_1x_1 and w_2x_2, then we are in case b). Otherwise, this means that there is a mismatch between the words $w_1x_1y_1^j$ and $w_2x_2y_2^j$, and we can unfold the loops to build two runs $\xrightarrow{w_1} p_1 \xrightarrow{uv^j|x_1y_1^j} q_1 \xrightarrow{v|y_1} q_1$ and $\xrightarrow{w_2} p_2 \xrightarrow{uv^j|x_2y_2^j} q_2 \xrightarrow{v|y_2} q_2$, and we are in case b).

It remains to prove that the two runs in case b) are always $2M_{\mathcal{T}}(|Q|^2+1)$-close. Suppose they are not. As a single transition can increase the length of an output word by at most $M_{\mathcal{T}}$, then the input word has length at least $|Q|^2$. This allows us to identify a synchronized loop in the two runs which increases the distance. But then we would be in case a), and this is a contradiction. □

From Lemma 3.10, we can derive a decision procedure in three phases. Let $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ be a finite-state transducer.

Phase 1 We non-deterministically guess a skeleton of a counter-example. This skeleton consists of the following informations:

— two pairs of states (p_1, p_2) and (q_1, q_2) in Q^2, with p_1 and p_2 initial states,

— whether case a) or case b) of Lemma 3.10 will be at fault.

Phase 2 We verify that there exists a loop in $\mathcal{T}^2$ around (q_1, q_2) representing two runs ρ and ρ' such that either $|\mathsf{out}(\rho)| \neq |\mathsf{out}(\rho')|$ if we are in case a) or $|\mathsf{out}(\rho)| = |\mathsf{out}(\rho')| \neq 0$ if we are in case b).

Note that, in both cases, we can prove by contradiction that we can find such a loop on an input word of length at most $2|Q|^2$. We let $L = 2M_{\mathcal{T}}|Q|^2$.

We build a directed graph with vertices in $Q^2 \times \{0, \ldots, L\}^2$. We add an edge from vertex (r_1, r_2, n_1, n_2) to vertex $(r_1', r_2', n_1+|w_1|, n_2+|w_2|)$ if there exist a letter $a \in A$ and some transitions $r_1 \xrightarrow{a|w_1} r_1'$ and $r_2 \xrightarrow{a|w_2} r_2'$ in $\mathcal{T}$. We then test if we can reach from vertex $(q_1, q_2, 0, 0)$ any vertex (q_1, q_2, n_1, n_2) such that either $n_1 \neq n_2$ if we are in case a) or $n_1 = n_2 \neq 0$ if we are in case b).

The size of the graph is in $O(|Q|^2 \times \log_2(L))$. As reachability in a graph can be decided in non-deterministic logarithmic space, we obtain that finding such a loop can also be done in NLogSpace.

Phase 3 We do this phase only for case b). In order to guess a mismatch between two synchronised runs whose outputs stay at a distance of at most $N = 2M_{\mathcal{T}}(|Q|^2+1)$, one can proceed as follows. We first build a directed graph with vertices in $Q^2 \times \mathbb{N}^{\leqslant N} \times (B \cup \{\bot\})^2$. The graph simulates pairs of runs ρ and ρ' on the same input word, and stores in its counter the distance between the outputs

of the two runs. Additionally, vertices allow to non-deterministically store the letter produced by the run which is ahead (ρ for instance), and then continue the simulation of ρ' until ρ' catches up ρ (*i.e.* the counter is equal to 0) and checks that the letter produced by ρ' is different from the one stored before. We then test if any vertex with a counter equal to 0 and having different stored letter is reachable. The size of the graph is in $O(|Q|^2 \times \log_2(N))$. Again, as reachability in a graph can be decided in non-deterministic logarithmic space, we obtain that finding such a mismatch can also be done in NLOGSPACE.

Each of these three phases can be done in NLOGSPACE, and we obtain an overall decision procedure in NLOGSPACE.

3.4 Sequentiality in Other Contexts

Automata can be viewed as functions from words to boolean values, thus describing languages. They can be represented as weighted automata over the Boolean semiring, *i.e.* $(\mathbb{B}, \vee, \wedge, \bot, \top)$. Similarly, transducers can be represented as weighted automata over the semiring of languages, *i.e.* $(\mathcal{P}(A^*), \cup, \cdot, \emptyset, \{\varepsilon\})$, where the concatenation $\cdot$ has been extended to languages.

As such, it is a legitimate question whether and how the notion of sequentiality applies in the larger setting of weighted automata. This is a subject of interest in many fiels and [LS06] provides an extensive survey. Let us recall some of the known results. The determinisability of weighted automata over a field is known to be decidable [LS06]. Also, the determinisability problem of weighted automata over the tropical semiring $(\mathbb{Z}, \min, +, +\infty, 0)$, despite being very challenging, is decidable for polynomially ambiguous automata [KL09; Kir12]. Finally, [FGR15] proved that the determinisability problem is decidable for functional weighted automata with set semantics over infinitary groups.

On a different matter, it is interesting to see how structural properties similar to the twinning property are used in many results. Here is a non-exhaustive list: a weak twinning property to decide multi-sequentiality for relational finite-state transducers [JF15], a twinning property of order k to decide the problem of the minimisation to k registers of copyful appending deterministic streaming string transducers [DRT16], a critical loop property to decide the realisability of multi-sequential specifications [EFJ18], ...

Chapter 4

k-Sequentiality of Finite-State Transducers

This chapter presents the work we developed in [Dav+17] to characterise, given $k \in \mathbb{N}$, the functional finite-state transducers that admit an equivalent k-sequential one. Whereas the initial publication was presented in a larger setting, namely weighted automata with set semantics over infinitary groups, we will, in the spirit of the rest of this thesis, restrict ourselves to functional finite-state transducers.

First recall that a finite-state transducer is k-sequential if it is the union of k sequential finite-state transducers. A finite-state transducer is multi-sequential if it is k-sequential for some $k \in \mathbb{N}$. Multi-sequential transducers have been studied in [CS86] where the authors devised a technique to characterise the functional finite-state transducers that admit an equivalent multi-sequential one, and more recently in [JF15] where this result was extended to relational finite-state transducers.

Going forward from these results, it is a natural question to ask whether the size of the union can be decreased. As explained in introduction, while evaluating a transducer with non-determinism present the risk to have unbounded number of parallel runs for the same input word, functional multi-sequential transducers can be evaluated using a thread for each member of the union and a join to collect the output of the only accepting run. Decreasing the size of the union then allows to reduce the number of threads required. This leads us to the problem of k-sequentiality which aims at deciding, given $k \in \mathbb{N}$, whether a functional finite-state transducer admits an equivalent k-sequential one.

We have seen in Section 2.4 that the class of functional k-sequential finite-state transducers is equivalent to the class of copyless appending deterministic streaming string transducers with k registers. Therefore, the problem of the k-sequentiality of functional finite-state transducers also solves the problem of the minimisation to k registers of copyless appending deterministic streaming string transducers.

In order to characterise the functional finite-state transducers that admit an equivalent k-sequential one we extend the work of [Cho77] around sequentiality. We devise a generalisation of the Lipschitz and twinning properties: a Lipshitz property of order k and a branching twinning property of order k. The informal idea for this generalisation is that if our function is realisable by a k-sequential transducer, then, when we consider $k+1$ inputs, the outputs corresponding to two of these inputs should remain close, with respect to their relative distance.

Note that [DRT16] also devises generalisations of Choffrut's work, to characterise the copyful appending deterministic streaming string transducers with k registers: a bounded-variation property of order k and a twinning property of order k. They operate differently and we will also highlight the differences.

4.1 Preliminaries

Definition 4.1 (delay)**.** Given $x, y \in B^*$, the *delay* between x and y is $x^{-1}y \in \mathcal{F}_B$. It is denoted by $\mathsf{delay}(x, y)$.

We first draw a link between the delay and the distance between two words.

Lemma 4.1. *For all words $x, y \in B^*$, $\mathsf{dist}_p(x, y) = |\mathsf{delay}(x, y)|$.*

Proof. Let $w = \mathsf{lcp}(x, y)$ and let $x', y' \in B^*$ such that $x = wx'$ and $y = wy'$. Then we have $\mathsf{dist}_p(x, y) = |x| + |y| - 2|\mathsf{lcp}(x, y)| = |wx'| + |wy'| - 2|w| = |x'| + |y'| = |(x')^{-1}| + |y'| = |(x')^{-1}y'| = |(x'w)^{-1}wy'| = |x^{-1}y| = |\mathsf{delay}(x, y)|$ □

The delay provides an additional tool to express the combinatorial constraint of the twinning property.

Lemma 4.2. *Let four words $x_1, x_2, y_1, y_2 \in B^*$. The following assertions are equivalent:*

1. *delay*$(x_1, x_2) =$ *delay*(x_1y_1, x_2y_2),
2. *there exists* $L \in \mathbb{N}$ *such that for all* $i \geqslant 0$, *dist*$_p(x_1y_1^i, x_2y_2^i) \leqslant L$,
3. *either* $y_1 = y_2 = \varepsilon$, *or* $|y_1| = |y_2|$ *and there exists* $z \in B^*$ *such that either* $x_1 = x_2z$ *and* $zy_1 = y_2z$, *or* $x_2 = x_1z$ *and* $zy_2 = y_1z$.

Proof. The equivalence between 2 and 3 has been proven in Section 3.1.3.

Let us first prove the implication from 1 to 2. We suppose that $\mathsf{delay}(x_1, x_2) = \mathsf{delay}(x_1y_1, x_2y_2)$, *i.e.* $x_1^{-1}x_2 = (x_1y_1)^{-1}x_2y_2 = y_1^{-1}x_1^{-1}x_2y_2$. We set $L = \mathsf{dist}_p(x_1, x_2)$. For all $i \geqslant 0$, we have $x_1^{-1}x_2 = (y_1^{-1})^i x_1^{-1}x_2y_2^i$. Therefore, $|x_1^{-1}x_2| = |(x_1y_1^i)^{-1}x_2y_2^i|$. By Lemma 4.1, we obtain that $\mathsf{dist}_p(x_1, x_2) = \mathsf{dist}_p(x_1y_1^i, x_2y_2^i)$.

To prove the implication from 3 to 1, we analyse the three cases. If $y_1 = y_2 = \varepsilon$ then the result is trivial. If $|y_1| = |y_2|$ and there exists $z \in B^*$ such that $x_1 = x_2z$ and $zy_1 = y_2z$ then we have $x_1y_1 = x_2y_2z$. Therefore, $\mathsf{delay}(x_1y_1, x_2y_2) = (x_1y_1)^{-1}x_2y_2 = (x_2y_2z)^{-1}x_2y_2 = z^{-1}$ and $\mathsf{delay}(x_1, x_2) = (x_1)^{-1}x_2 = (x_2z)^{-1}x_2 = z^{-1}$. The third case is symmetrical. □

We say that a function f is k-sequential if it can be realised by a k-sequential finite-state transducer. We now define the degree of sequentiality of a function.

Definition 4.2 (Degree of sequentiality)**.** The *degree of sequentiality* of a function f is the minimal $k \in \mathbb{N}$ such that f is k-sequential.

4.2 Characterisation of k-Sequential Functions

4.2.1 Lipschitz Property of Order k

We lift the Lipschitz property to functions that can be expressed using a k-sequential transducer: given a function f, we consider $k + 1$ input words and require that two of those must have proportionally close images by f.

Definition 4.3 (Lipschitz property of order k)**.** Let A, B be two alphabets. A function f from A^* to B^* satisfies the *Lipschitz property of order* k if there exists $K \in \mathbb{N}$ such that for all $u_0, \ldots, u_k \in \mathsf{dom}(f)$, there exist two indices i, j such that $0 \leqslant i < j \leqslant k$ and $\mathsf{dist}_p(f(u_i), f(u_j)) \leqslant K.\mathsf{dist}_p(u_i, u_j)$.

Remark. Note that the Lipschitz property introduced in Definition 3.2 is equivalent to the Lipschitz property of order 1. Also, it must be pointed out that we use a generalisation of the Lipschitz property whereas [DRT16] used a generalisation of the bounded variation property.

Example 4.1. As shown in Chapter 3, the function f_{last} defined in Example 1.3 does not satisfy the Lipschitz property, and thus it does not satisfy the Lipschitz property of order 1. One can however prove that this function satisfies the Lipschitz property of order 2.

4.2.2 Branching Twinning Property of Order k

The idea behind the branching twinning property of order k is to consider $k+1$ runs labeled by arbitrary words with k cycles. If the branching twinning property is satisfied then there are two runs among these $k+1$ such that the outputs remain close (i.e. the prefix distance between these values is bounded) along the prefix part of these two runs that read the same input. This property is named after the intuition that the $k+1$ runs can be organized in a tree structure where the prefixes of any two runs are on the same branch up to the point where those two runs do not read the same input anymore.

While the twinning property of order k of [DRT16] simply goes from 2 to $k+1$ runs on the same input word, the branching twinning property of order k considers runs on inputs that may be different. Observe that the branching twinning property of order k is thus a strengthening of the twinning property of order k.

Definition 4.4 (Branching twinning property of order k)**.** Let A, B be two alphabets. A functional FST from A^* to B^* satisfies the *branching twinning property of order* k (denoted by BTP_k) if (see Figure 4.1)

— for all states $q_{i,j}$ with $0 \leqslant i \leqslant k, 0 \leqslant j \leqslant k$ and $q_{0,j}$ initial for all $0 \leqslant j \leqslant k$,

— for all words $u_{i,j}, v_{i,j} \in A^*$ with $1 \leqslant i \leqslant k$ and $0 \leqslant j \leqslant k$

such that there are $k+1$ runs satisfying

— $\xrightarrow{w_j} q_{0,j}$ for all $0 \leqslant j \leqslant k$, and

— $q_{i-1,j} \xrightarrow{u_{i,j}|x_{i,j}} q_{i,j}$ and $q_{i,j} \xrightarrow{v_{i,j}|y_{i,j}} q_{i,j}$ for all $1 \leqslant i \leqslant k, 0 \leqslant j \leqslant k$,

there exists $0 \leqslant j < j' \leqslant k$ such that for all $1 \leqslant i \leqslant k$, if for every $1 \leqslant i' \leqslant i$, we have $u_{i',j} = u_{i',j'}$ and $v_{i',j} = v_{i',j'}$, then we have

$$\mathsf{delay}(w_j x_{1,j} \cdots x_{i,j}, w_{j'} x_{1,j'} \cdots x_{i,j'}) = \mathsf{delay}(w_j x_{1,j} \cdots x_{i,j} y_{i,j}, w_{j'} x_{1,j'} \cdots x_{i,j'} y_{i,j'}).$$

Remark. Note that the twinning property introduced in Chapter 3 is equivalent to the branching twinning property of order k for $k = 1$.

Example 4.2. As shown in Chapter 3, the finite-state transducer $\mathcal{T}_{last}$ defined in Example 1.3 does not satisfy the twinning property. We can show using the same counter-example that it also does not satisfy the BTP_1. Indeed, consider two loops around q_a and q_b: $q_{0,0} = q_{1,0} = q_a$, $q_{0,1} = q_{1,1} = q_b$, $u_{1,0} = u_{1,1} = \varepsilon$ and $v_{1,0} = v_{1,1} = \mathsf{a}$. Then, $\mathsf{delay}(\varepsilon, \varepsilon) = \varepsilon \neq \mathsf{delay}(\mathsf{a}, \mathsf{b}) = \mathsf{a}^{-1}\mathsf{b}$. One can prove however that it satisfies the BTP_2. Therefore, the sequentiality degree of f_{last} is 2.

Figure 4.2 depicts $\mathcal{T}_{last^2}$, the finite-state transducer obtained by concatenating $\mathcal{T}_{last}$ with itself, with a fresh $\#$ separator letter. $\mathcal{T}_{last^2}$ realizes the function $f_{last^2} : u\#v \mapsto f_{last}(u)\#f_{last}(v)$ where $u, v \in \{a, b\}^+$. We can see that the minimal k such that $\mathcal{T}_{last^2}$ satisfies the BTP_k is $k = 4$. Therefore, the sequentiality degree of f_{last^2} is 4.

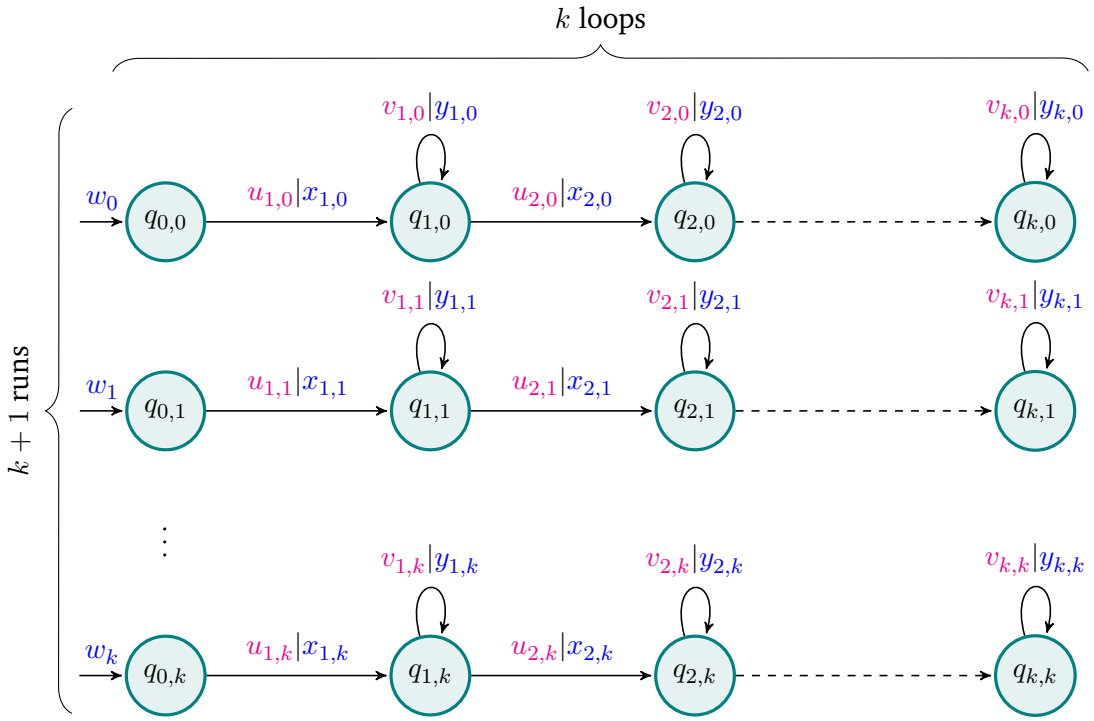


Figure 4.1 – Branching twinning property of order k

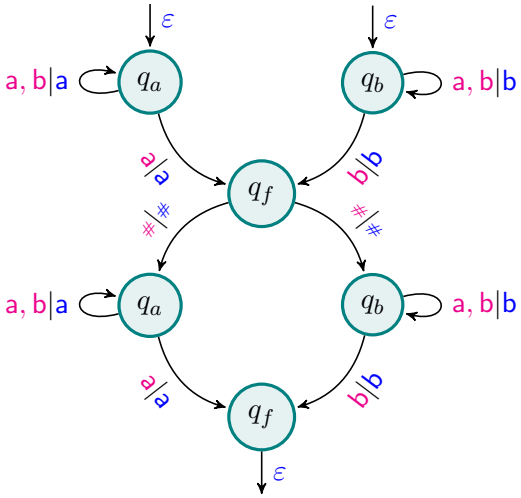


Figure 4.2 – The FST $\mathcal{T}_{last^2}$.

4.2.3 k-Sequentialisation Theorem

Our main result is the following theorem, which characterises the functional finite-state transducers admitting an equivalent k-sequential finite-state transducer.

Theorem 4.3. *Let A, B be two alphabets. Let $\mathcal{T}$ be a functional FST from A^* to B^**

and let $k \in \mathbb{N}$. The following assertions are equivalent:

1. *$[\![\mathcal{T}]\!]$ satisfies the Lipschitz property of order k,*
2. *$\mathcal{T}$ satisfies the branching twinning property of order k,*
3. *$[\![\mathcal{T}]\!]$ can be realised by a k-sequential FST.*

Proof. The implications from 3 to 1 and from 1 to 2 are proved in Propositions 4.4 and 4.5. The implication from 2 to 3 involves the construction of an equivalent k-sequential FST which is detailed and proved in Section 4.3. □

Proposition 4.4. *Let A, B be two alphabets and let $k \in \mathbb{N}$. Let $\mathcal{T}$ be a k-sequential functional FST realising the function f from A^* to B^*. Then f satisfies the Lipschitz property of order k.*

Proof. Consider that $\mathcal{T}$ is defined as the union of k sequential FSTs $\mathcal{T}_1, \dots, \mathcal{T}_k$. Let $u_0, \dots, u_k \in \mathsf{dom}([\![\mathcal{T}]\!])$. By the pigeon hole principle, there are $0 \leqslant j < j' \leqslant k$ and $1 \leqslant i \leqslant k$ such that $u_j, u_{j'} \in \mathsf{dom}([\![\mathcal{T}_i]\!])$. The result follows by sequentiality of $\mathcal{T}_i$ (*cf.* Proposition 3.6). □

Proposition 4.5. *Let A, B be two alphabets. Let $\mathcal{T}$ be a functional FST realising the function f from A^* to B^* and let $k \in \mathbb{N}$. If f satisfies the Lipschitz property of order k, then $\mathcal{T}$ satisfies the branching twinning property of order k.*

Consider a functional finite-state transducer $\mathcal{T}$ that does not satisfy BTP_k. Let us prove that $[\![\mathcal{T}]\!]$ does not satisfy Lip_k. It is a consequence of the following lemma.

Lemma 4.6. *If $\mathcal{T}$ does not satisfy BTP_k, then for all positive integers K, there are $k+1$ words $u_0, \dots, u_k$, initial states $q_0, \dots, q_k$, states $p_0, \dots, p_k$ and $k+1$ runs:*

$$\xrightarrow{w_j} q_j \xrightarrow{u_j|x_j} p_j \quad \textit{for all } 0 \leqslant j \leqslant k,$$

such that for all $j \neq j'$, $\mathsf{dist}_p(w_j x_j, w_{j'} x_{j'}) \geqslant K.\max(\mathsf{dist}_p(u_j, u_{j'}), 1)$.

Proof. The idea behind the proof is to consider a witness as described in Figure 4.1. If BTP_k is not satisfied, then one can pump the loops "the right number of times" to: (1) sufficiently increase the prefix distance between the outputs of the runs, (2) not increase too much the distance between the corresponding input words.

Let K be a positive integer. Since $\mathcal{T}$ does not satisfy BTP_k, then there are:
— states $q_{i,j}$ with $0 \leqslant i \leqslant k, 0 \leqslant j \leqslant k$ and $q_{0,j}$ initial for all $0 \leqslant j \leqslant k$,
— words $u_{i,j}$ and $v_{i,j}$ with $1 \leqslant i \leqslant k$ and $0 \leqslant j \leqslant k$, and
— $k+1$ runs such that
  — $\xrightarrow{w_j} q_{0,j}$ for all $0 \leqslant j \leqslant k$, and
  — $q_{i-1,j} \xrightarrow{u_{i,j}|x_{i,j}} q_{i,j}$ and $q_{i,j} \xrightarrow{v_{i,j}|y_{i,j}} q_{i,j}$ for all $1 \leqslant i \leqslant k, 0 \leqslant j \leqslant k$,

such that for all $0 \leqslant j < j' \leqslant k$, there is $1 \leqslant i \leqslant k$ such that for all $1 \leqslant i' \leqslant i$, we have $u_{i',j} = u_{i',j'}$, $v_{i',j} = v_{i',j'}$ and

$$\mathsf{delay}(w_j x_{1,j} \cdots x_{i,j}, w_{j'} x_{1,j'} \cdots x_{i,j'}) \neq \mathsf{delay}(w_j x_{1,j} \cdots x_{i,j} y_{i,j}, w_{j'} x_{1,j'} \cdots x_{i,j'} y_{i,j'}).$$

We construct by induction (in decreasing order) a sequence of positive integers $t_k, \ldots, t_1$. Let us give the construction of t_i, assuming $t_{i+1}, \ldots, t_k$ have been defined. Let L'_i be the maximal length of the words $u_{i+1,j} v_{i+1,j}^{t_{i+1}} \cdots u_{k,j} v_{k,j}^{t_k}$ over all $0 \leqslant j \leqslant k$, and let $L_i = \max(L'_i, 1)$. Consider J_i the set of pairs (j, j') such that for all $1 \leqslant i' \leqslant i$, we have $u_{i',j} = u_{i',j'}$, $v_{i',j} = v_{i',j'}$ and

$$\mathsf{delay}(w_j x_{1,j} \cdots x_{i,j}, w_{j'} x_{1,j'} \cdots x_{i,j'}) \neq \mathsf{delay}(w_j x_{1,j} \cdots x_{i,j} y_{i,j}, w_{j'} x_{1,j'} \cdots x_{i,j'} y_{i,j'}).$$

By Lemma 4.2, one can choose an integer N such that for all pairs $(j, j') \in J_i$,

$$\mathsf{dist}_p(w_j x_{1,j} \cdots x_{i,j} (y_{i,j})^N, w_{j'} x_{1,j'} \cdots x_{i,j'} (y_{i,j'})^N) \geqslant 2L_i(M_{\mathcal{T}} + K). \qquad (*)$$

We set $t_i = N$.

We show that the words $u_j = u_{1,j} v_{1,j}^{t_1} \cdots u_{k,j} v_{k,j}^{t_k}$, for all $0 \leqslant j \leqslant k$, and the corresponding runs fulfil the condition of the lemma. Indeed, let $j \neq j'$, and i the minimal index such that $(j, j') \in J_i$. Such an index i exists by hypothesis. For $\ell \in \{j, j'\}$, set

$$\begin{aligned}
x_\ell &= x_{1,\ell} (y_{1,\ell})^{t_1} x_{2,\ell} (y_{2,\ell})^{t_2} \cdots x_{k,\ell} (y_{k,\ell})^{t_k} \text{ and} \\
\overline{x}_\ell &= x_{1,\ell} x_{2,\ell} \cdots x_{i-1,\ell} x_{i,\ell} (y_{i,\ell})^{t_i} \text{ and} \\
\overline{\overline{x}}_\ell &= x_{i+1,\ell} (y_{i+1,\ell})^{t_{i+1}} \cdots x_{k,\ell} (y_{k,\ell})^{t_k}.
\end{aligned}$$

Because i is the minimal index such that $(j, j') \in J_i$, there is no delay induced by the loops up to the i^{th} loop and we obtain

$$\mathsf{delay}(w_j x_j, w_{j'} x_{j'}) = \mathsf{delay}(w_j \overline{x}_j \overline{\overline{x}}_j, w_{j'} \overline{x}_{j'} \overline{\overline{x}}_{j'}) = \overline{\overline{x}}_j^{-1} \mathsf{delay}(w_j \overline{x}_j, w_{j'} \overline{x}_{j'}) \overline{\overline{x}}_{j'}$$

Then, by Lemma 4.1,

$$\begin{aligned}
\mathsf{dist}_p(w_j x_j, w_{j'} x_{j'}) &= |\mathsf{delay}(w_j x_j, w_{j'} x_{j'})| \\
&\geqslant |\mathsf{delay}(w_j \overline{x}_j, w_{j'} \overline{x}_{j'})| - |\overline{\overline{x}}_j^{-1}| - |\overline{\overline{x}}_{j'}|
\end{aligned}$$

By definition of L_i, $|\overline{\overline{x}}_j^{-1}| \leqslant L_i M_{\mathcal{T}}$ and $|\overline{\overline{x}}_{j'}| \leqslant L_i M_{\mathcal{T}}$, and then by (*)

$$\mathsf{dist}_p(w_j x_j, w_{j'} x_{j'}) \geqslant 2L_i(M_{\mathcal{T}} + K) - 2L_i M_{\mathcal{T}} \geqslant 2L_i K$$

Moreover, by definition of L_i, $\mathsf{dist}_p(u_j, u_{j'}) \leqslant 2L_i$ and $L_i \geqslant 1$.

Therefore

$$\mathsf{dist}_p(w_j x_j, w_{j'} x_{j'}) \geqslant K.\max(\mathsf{dist}_p(u_j, u_{j'}), 1) \qquad \square$$

Proof of Proposition 4.5. We prove that if $\mathcal{T}$ does not satisfy BTP_k then $[\![\mathcal{T}]\!]$ does not satisfy Lip_k. Let $K \in \mathbb{N}$ and $K' = K(2n+1) + 2(n+1)M_{\mathcal{T}}$ where n is the number of states of $\mathcal{T}$. By Lemma 4.6, there are $k+1$ words $u_0, \ldots, u_k$, initial states $q_0, \ldots, q_k$, states $p_0, \ldots, p_k$ and $k+1$ runs:

$$\xrightarrow{w_j} q_j \xrightarrow{u_j|x_j} p_j \quad \text{for all } 0 \leqslant j \leqslant k,$$

such that for all $j \neq j'$, $\mathsf{dist}_p(w_j x_j, w_{j'} x_{j'}) \geqslant K'.\max(\mathsf{dist}_p(u_j, u_{j'}), 1)$.

As $\mathcal{T}$ is trimmed, these $k+1$ runs can be completed into accepting runs:

$$\xrightarrow{w_j} q_j \xrightarrow{u_j|x_j} p_j \xrightarrow{v_j|y_j} r_j \xrightarrow{z_j} \quad \text{for all } 0 \leqslant j \leqslant k,$$

such that for all $0 \leqslant j \leqslant k$, $|v_j| \leqslant n$. Then, for all $j \neq j'$, we have

$$\begin{aligned}
&\mathsf{dist}_p(w_j x_j y_j z_j, w_{j'} x_{j'} y_{j'} z_{j'}) \\
&\quad \geqslant \mathsf{dist}_p(w_j x_j, w_{j'} x_{j'}) - 2(n+1)M_{\mathcal{T}} && \text{by the triangle inequality twice} \\
&\quad \geqslant K'.\max(\mathsf{dist}_p(u_j, u_{j'}), 1) - 2(n+1)M_{\mathcal{T}} && \text{by Lemma 4.6} \\
&\quad \geqslant (K(2n+1) + 2(n+1)M_{\mathcal{T}}).\max(\mathsf{dist}_p(u_j, u_{j'}), 1) - 2(n+1)M_{\mathcal{T}} \\
&\quad \geqslant K(2n+1).\max(\mathsf{dist}_p(u_j, u_{j'}), 1) && \text{because } \max(\mathsf{dist}_p(u_j, u_{j'}), 1) \geqslant 1 \\
&\quad \geqslant K(\mathsf{dist}_p(u_j, u_{j'}) + 2n) && \text{idem} \\
&\quad \geqslant K.\mathsf{dist}_p(u_j v_j, u_{j'} v_{j'}) && \square
\end{aligned}$$

4.3 Construction of a k-Sequential Equivalent

In this section, we consider a functional FST $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ that satisfies the BTP_k. We will build a transducer $\mathcal{D}$ that is equivalent to $\mathcal{T}$ defined as the union of k sequential transducers $\mathcal{D}_i$, for $1 \leqslant i \leqslant k$.

4.3.1 An Infinite Sequential Equivalent

We first build $\mathcal{D}_\omega = (Q', t'_{\mathit{init}}, t'_{\mathit{final}}, T')$ an infinite sequential transducer equivalent to $\mathcal{T}$, using the construction of Section 3.2. We recall that this construction was a power set construction extended with unproduced output words. In particular, we have $Q' \subseteq 2^{Q \times B^*}$. We will only consider the accessible part of $\mathcal{D}_\omega$.

4.3.2 Recovering k-Sequentiality

We let $N_{\mathcal{T}} = 2M_{\mathcal{T}}|Q|^{|Q|}$. The states of $\mathcal{D}_\omega$ are sets of pairs of a state of $\mathcal{T}$ and a word. As such, they can be viewed as partial functions from states of $\mathcal{T}$ to words. In the following lemma, we will prove that when a state of $\mathcal{D}_\omega$ contains pairs with words that are too far apart (their relative distance is more than $N_{\mathcal{T}}$) then a copy of $\mathcal{T}$ of which we replace the initial function by this state is a k-sequential transducer. This will allow us to define $\mathcal{D}$ from $\mathcal{D}_\omega$.

Lemma 4.7. *For all $S \in Q'$, if there exists $(q_1, w_1), (q_2, w_2) \in S$ such that $\mathit{dist}_p(w_1, w_2) > N_{\mathcal{T}}$, then $\mathcal{T}_S$ is k-sequential.*

Definition 4.5. For a state $S \in Q'$, we define the *rank* of S, denoted by $\mathsf{rank}(S)$, as the minimal integer k' such that $\mathcal{T}_S$ satisfies the $\mathsf{BTP}_{k'}$.

The input word of a run ρ is denoted by $in(\rho)$. If a run ρ in $\mathcal{T}$ has length at least the number of states of Q then, by the pigeon hole principle, ρ contains a loop. Using this idea, the following lemma can be proven by induction.

Lemma 4.8. *Let ρ be a run in $\mathcal{T}$. If $|in(\rho)| \geqslant |Q|$ then there exist $\ell \geqslant 1$, and runs $\rho_0, \dots, \rho_\ell$ and $\rho'_1, \dots, \rho'_\ell$, such that $\rho = \rho_0\rho'_1\rho_1 \dots \rho'_\ell\rho_\ell$, all the ρ'_i are loops and $|in(\rho_0 \dots \rho_\ell)| < |Q|$.*

Lemma 4.9. *For all $S \in Q'$, if there exist $(q_1, w_1), (q_2, w_2) \in S$ such that $\mathit{dist}_p(w_1, w_2) > N_{\mathcal{T}}$, then there exists a partition of S in two subsets S' and S'' such that $\mathsf{rank}(S') + \mathsf{rank}(S'') \leqslant k$.*

Proof. Let $(q_1, w_1), (q_2, w_2) \in S$ such that $\mathsf{dist}_p(w_1, w_2) > N_{\mathcal{T}}$. Let $(q_3, w_3), \dots, (q_m, w_m)$ be an enumeration of the elements of S distinct from (q_1, w_1) and (q_2, w_2). Because $\mathcal{T}$ is functional and $\mathcal{D}_\omega$ is equivalent to $\mathcal{T}$, we have $m \leqslant |Q|$.

Since S is accessible, there exists a run $\xrightarrow[\mathcal{D}_\omega]{s} I \xrightarrow[\mathcal{D}_\omega]{u|t} S$. Then, by (P2) of Section 3.2, for every $1 \leqslant j \leqslant m$, since $(q_j, w_j) \in S$, there exists a run

$$\rho_j : \xrightarrow[\mathcal{T}]{x_{0,j}} q_{0,j} \xrightarrow[\mathcal{T}]{u|x_j} q_j$$

such that $stw_j = x_{0,j}x_j$.

The proof is now done in two steps. First, we expose a decomposition of u into three words $wv_\tau w'$ such that every run ρ_j loops over v_τ, and the delay between the outputs of the runs ρ_1 and ρ_2 is modified along v_τ. This allows us to define a partition $\{S', S''\}$ of S, splitting the elements (q_j, w_j) of S depending on whether or not the delay between ρ_1 and ρ_j changes along v_τ. Then, we prove that $\mathsf{rank}(S') + \mathsf{rank}(S'') \leqslant k$, using the following idea. Let $r' = \mathsf{rank}(S') - 1$, $r'' = \mathsf{rank}(S'') - 1$. By combining a witness of the non satisfaction of the $\mathsf{BTP}_{r'}$ by $\mathcal{T}_{S'}$ and a witness of the non satisfaction of the $\mathsf{BTP}_{r''}$ by $\mathcal{T}_{S''}$, we build a witness of the non satisfaction of the $\mathsf{BTP}_{r'+r''+1}$ by $\mathcal{T}_S$. This in turn implies that

$$\mathsf{rank}(S') + \mathsf{rank}(S'') - 1 = r' + r'' + 1 < k.$$

1. By applying Lemma 4.8 to the product of m copies of $\mathcal{T}$, there exists $\ell \geqslant 1$ such that we obtain a subdivision $u_0v_1u_1 \cdots v_\ell u_\ell$ of the word u such that each run ρ_j loops over each input v_s, and $|u_0 \cdots u_\ell| < |Q|^m \leqslant |Q|^{|Q|}$.
   Note that the distance between the outputs of ρ_1 and ρ_2 after reading the input u is
   $$\mathsf{dist}_p(w_1, w_2) > N_{\mathcal{T}} = 2M_{\mathcal{T}}|Q|^{|Q|}.$$
   In order for the distance to increase by at least $N_{\mathcal{T}}$, there has to exist an integer $1 \leqslant \tau \leqslant \ell$ such that the delay between the outputs of ρ_1 and ρ_2 changes along v_τ, since $M_{\mathcal{T}}$ is greater than or equal to the maximal size of the output of a transition of $\mathcal{T}$, and $|u_0 \cdots u_\ell| < |Q|^{|Q|}$. Let $\overline{u} = u_0v_1u_1 \cdots v_{\tau-1}u_{\tau-1}$, $\overline{\overline{u}} = u_\tau v_{\tau+1}u_{\tau+1} \cdots v_\ell u_\ell$, and for every $1 \leqslant j \leqslant m$, consider the following decomposition of the run ρ_j:
   $$\rho_j : \xrightarrow{x_{0,j}} q_{0,j} \xrightarrow{\overline{u}|x_{1,j}} q_{1,j} \xrightarrow{v_\tau|x_{2,j}} q_{1,j} \xrightarrow{\overline{\overline{u}}|x_{3,j}} q_j.$$
   Let S' be the set of pairs (q_j, w_j) corresponding to the indices j such that the delay between ρ_1 and ρ_j stays the same along v_τ, *i.e.* ,
   $$\mathsf{delay}(x_{0,1}x_{1,1}, x_{0,j}x_{1,j}) = \mathsf{delay}(x_{0,1}x_{1,1}x_{2,1}, x_{0,j}x_{1,j}x_{2,j}),$$
   Then, let $S'' = S \setminus S'$. By definition of v_τ, S'' is not empty since it contains (q_2, w_2), hence $\{S', S''\}$ is a partition of S.
2. Let $r' = \mathsf{rank}(S') - 1$, $r'' = \mathsf{rank}(S'') - 1$, and $r = r' + r''$. By definition of the rank, there exists an unsatisfied instance of the $\mathsf{BTP}_{r'}$ over the transducer $\mathcal{T}_{S'}$. Let $\phi_0, \ldots, \phi_{r'}$ be the runs of $\mathcal{T}_{S'}$ forming this instance. Similarly, there exists an unsatisfied instance of the $\mathsf{BTP}_{r''}$ over the transducer $\mathcal{T}_{S''}$. Let $\phi_{r'+1}, \ldots, \phi_{r+1}$ be the runs of $\mathcal{T}_{S''}$ forming this instance. For every $0 \leqslant i \leqslant r+1$, we add loops over the empty word at the end of the run ϕ_i in order for it to contain exactly r loops, yielding the run:
   $$\phi'_i : \xrightarrow{y_{0,i}} p_{0,i} \xrightarrow{s_{1,i}|y_{1,i}} p_{1,i} \overset{t_{1,i}|z_{1,i}}{\circlearrowright} \cdots\cdots \overset{t_{r,i}|z_{r,i}}{\circlearrowright} p_{r,i}$$
   Note that, since both S' and S'' are subsets of S, each ϕ'_i can be seen as a run over the transducer $\mathcal{T}_S$. By extending those runs on the left, we now construct an instance of the BTP_{r+1} for $\mathcal{T}$ that is not satisfied. For every $0 \leqslant i \leqslant r+1$, $(p_{0,i}, y_{0,i}) \in S$, therefore there exists $1 \leqslant j_i \leqslant m$ such that $(p_{0,i}, y_{0,i}) = (q_{j_i}, w_{j_i})$, hence we can compose the run ρ_{j_i} with ϕ'_i, as follows:
   $$\psi_i : \xrightarrow{x_{0,j_i}} q_{0,j_i} \xrightarrow{\overline{u}|x_{1,j_i}} q_{1,j_i} \overset{v_\tau|x_{2,j_i}}{\circlearrowright} \xrightarrow{\overline{\overline{u}}|x_{3,j_i}} q_{j_i} \xrightarrow{s_{1,i}|y_{1,i}} p_{1,i} \overset{t_{1,i}|z_{1,i}}{\circlearrowright} \cdots\cdots \overset{t_{r,i}|z_{r,i}}{\circlearrowright} p_{r,i}$$

Note that ψ_i is a run of $\mathcal{T}$.

In order to conclude the proof, we need to prove that the instance of the BTP_{r+1} for $\mathcal{T}$ formed by the runs $\psi_0, \ldots, \psi_r$ is not satisfied. For every $0 \leqslant i < i' \leqslant r+1$, we now expose a loop differentiating ψ_i and $\psi_{i'}$, *i.e.* , a loop along which the delay between the outputs of ψ_i and $\psi_{i'}$ changes, and that occurs on the part of the runs where the inputs are identical.

We consider three possibilities:

— If $0 \leqslant i \leqslant r' < i' \leqslant r+1$, then $(q_{j_i}, w_{j_i}) \in S'$ and $(q_{j_{i'}}, w_{j_{i'}}) \in S''$, and, by definition of the partition $\{S', S''\}$ of S,

$$\mathsf{delay}(x_{0,j_i}x_{1,j_i}, x_{0,j_{i'}}x_{1,j_{i'}}) \neq \mathsf{delay}(x_{0,j_i}x_{1,j_i}x_{2,j_i}, x_{0,j_{i'}}x_{1,j_{i'}}x_{2,j_{i'}}),$$

Therefore, the first loop of ψ_i and $\psi_{i'}$ can be used to differentiate them.

— If $0 \leqslant i < i' \leqslant r'$, since the runs $\phi_0, \ldots, \phi_{r'}$ form a non satisfied instance of the $\mathsf{BTP}_{r'}$ for $\mathcal{T}_{S'}$, we can find a loop differentiating ϕ_i and $\phi_{i'}$, and use the corresponding loop to differentiate ψ_i and $\psi_{i'}$.

— If $r'+1 \leqslant i < i' \leqslant r+1$, since the runs $\phi_{r'+1}, \ldots, \phi_{r+1}$ form a non satisfied instance of the $\mathsf{BTP}_{r''}$ for $\mathcal{T}_{S''}$, we can find a loop differentiating ϕ_i and $\phi_{i'}$, and use the corresponding loop to differentiate ψ_i and $\psi_{i'}$.

As $\mathcal{T}$ satisfies the BTP_k, so does $\mathcal{T}_S$ and we have $\mathsf{rank}(S) \leqslant k$. As $\mathcal{T}_S$ does not satisfy the $\mathsf{BTP}_{r'+r''+1}$, we obtain that $\mathsf{rank}(S')+\mathsf{rank}(S'')-1 = r'+r''+1 < \mathsf{rank}(S)$, and thus $\mathsf{rank}(S') + \mathsf{rank}(S'') \leqslant k$. □

Proof of Lemma 4.7. By Lemma 4.9, there exists a partition of S into two subsets S' and S'' such that $\mathsf{rank}(S') + \mathsf{rank}(S'') \leqslant k$. Note that $\mathsf{rank}(S') \geqslant 1$ and $\mathsf{rank}(S'') \geqslant 1$, hence $\mathsf{rank}(S'') < k$ and $\mathsf{rank}(S') < k$. Therefore, the induction hypothesis can be applied, proving that $\mathcal{T}_{S'}$ is $\mathsf{rank}(S')$-sequential, and $\mathcal{T}_{S''}$ is $\mathsf{rank}(S'')$-sequential. Finally, as S is equal to the union of S' and S'', $\mathcal{T}_S$ is equivalent to the union of $\mathcal{T}_{S'}$ and $\mathcal{T}_{S''}$. Therefore $\mathcal{T}_S$ is k-sequential. □

4.3.3 Building a k-Sequential

We build k sequential transducers $\mathcal{D}_1, \ldots, \mathcal{D}_k$ whose union is equivalent to $\mathcal{T}$. Let U denote the set containing the accessible states S of $\mathcal{D}_\omega$ such that for all $(q_1, w_1), (q_2, w_2) \in S$, $\mathsf{dist}_p(w_1, w_2) \leqslant N_\mathcal{T}$. Moreover, let U' be the set of states of $\mathcal{D}_\omega$ accessible in one transition from U, *i.e.* ,

$$U' = \{S' \mid \exists S \in U, a \in A, w \in B^* \text{ such that } (S, a, w, S') \in T'\}.$$

There are only finitely many $w_1, w_2 \in B^*$ such that $\mathsf{dist}_p(w_1, w_2) \leqslant N_\mathcal{T}$ and $\mathsf{lcp}(w_1, w_2) = \varepsilon$. Hence U is finite. Note that this implies the finiteness of U'.

By Lemma 4.7, for every state $S \in U'$ that is not in U, $\mathcal{T}_S$ can be expressed as the union of k sequential transducers $\mathcal{D}_{S,i}$, with $1 \leqslant i \leqslant k$. For every $1 \leqslant i \leqslant k$,

let $\mathcal{D}_i$ be defined as the union of $\mathcal{D}_\omega$ restricted to the states $S \in U$, and all the $\mathcal{D}_{S',i}$, for $S' \in U' \setminus U$, with the two following differences:

1. the only initial state of $\mathcal{D}_i$ is the initial state of $\mathcal{D}_\omega$,
2. for every transition (S, a, w, S') of $\mathcal{D}_\omega$ between states $S \in U$ and $S' \in U' \setminus U$, we add a transition $(S, a, ww_{S',i}, q_{S',i})$, where $\xrightarrow[\mathcal{D}_{S',i}]{w_{S',i}} q_{S',i}$.

Proposition 4.10. *$\mathcal{D} = \cup_{1 \leqslant i \leqslant k} \mathcal{D}_i$ is a finite k-sequential FST equivalent to $\mathcal{T}$.*

Proof. We prove that $\mathcal{D}_\omega$ is equivalent to the union of the $\mathcal{D}_i$, with $1 \leqslant i \leqslant k$, which implies the desired result, since $\mathcal{T}$ is equivalent to $\mathcal{D}_\omega$ and the $\mathcal{D}_i$ are finite by construction.

First, we show that $[\![\mathcal{D}_\omega]\!]$ is included into the union of the $[\![\mathcal{D}_i]\!]$. Let $\rho : \xrightarrow{w} S_0 \xrightarrow{u|x} S_f \xrightarrow{y}$ be an accepting run of $\mathcal{D}_\omega$. We now expose an integer $i \in \{1, \ldots, k\}$ such that the output of the run of $\mathcal{D}_i$ over the input u is wxy. If all the states visited by ρ are in U, ρ is present in each $\mathcal{D}_i$, and we are done. Otherwise, let us split ρ as follows:

$$\rho : \xrightarrow{w} S_0 \xrightarrow{u_1|x_1} S \xrightarrow{a|x'} S' \xrightarrow{u_2|x_2} S_f \xrightarrow{y},$$

where S' is the first state encountered along ρ that is not in U. Then $S' \in U'$. Moreover, by (P2) of Section 3.2, the definition of the final relation of $\mathcal{D}_\omega$, and the fact that $\mathcal{T}_{S'}$ is equivalent to the union of the $\mathcal{D}_{S',i}$, there exists $1 \leqslant i \leqslant k$ and a run

$$\rho' : \xrightarrow{z_0} q_0 \xrightarrow{u_2|z_1} q_f \xrightarrow{z_2}$$

in $\mathcal{D}_{S',i}$ such that $z_0 z_1 z_2 = x_2 y$. Then the run

$$\xrightarrow{w} S_0 \xrightarrow{u_1|x_1} S \xrightarrow{a|x'z_0} q_0 \xrightarrow{u_2|z_1} q_f \xrightarrow{z_2}$$

over the input u is in $\mathcal{D}_i$, and the associated output is $wx_1x'z_0z_1z_2 = wxy$, which proves the desired result.

Conversely, we can prove, using similar arguments, that for every $1 \leqslant i \leqslant k$, $[\![\mathcal{D}_i]\!]$ is included into $[\![\mathcal{D}_\omega]\!]$, which concludes the proof. □

4.4 Deciding k-Sequentiality

In this section, we prove the decidability of the following problem:

Problem 4.1 (k-sequentiality)**.** Given a functional FST $\mathcal{T}$ from A^* to B^* and a natural number k, is $[\![\mathcal{T}]\!]$ k-sequential?

Thanks to Theorem 4.3, deciding k-sequentiality is equivalent to deciding the branching twinning property of order k:

Problem 4.2 (BTP_k)**.** Given a functional FST $\mathcal{T}$ from A^* to B^* and a natural number k, does $\mathcal{T}$ satisfy the BTP_k?

We will now devise a decision procedure that proceeds similarly to the one we developed in Section 3.3. Given a functional FST and a natural number k, this procedure will non-deterministically find a counter-example to the branching twinning property of order k. Contrarily to Section 3.3, the counter-example here has $k+1$ runs with k loops.

We first show, when given a counter-example for the BTP_k, that, for each pair of run indices (j, j'), checking the delay constraint is equivalent to looking for a loop whose output words have distinct lengths, or for a mismatch on the paths leading to loops. The proof goes along the same line as the one of Lemma 3.10.

Lemma 4.11. *Let $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ be an FST. $\mathcal{T}$ violates the BTP_k if and only if there exists a counter example given as follows: there are*

- *states $q_{i,j}$ with $0 \leqslant i \leqslant k$, $0 \leqslant j \leqslant k$ and $q_{0,j}$ initial for all $0 \leqslant j \leqslant k$,*
- *words $u_{i,j}$ and $v_{i,j}$ with $1 \leqslant i \leqslant k$ and $0 \leqslant j \leqslant k$, and*
- *$k+1$ runs such that*
  - *$\xrightarrow{w_j} q_{0,j}$ for all $0 \leqslant j \leqslant k$, and*
  - *$q_{i-1,j} \xrightarrow{u_{i,j}|x_{i,j}} q_{i,j}$ and $q_{i,j} \xrightarrow{v_{i,j}|y_{i,j}} q_{i,j}$ for all $1 \leqslant i \leqslant k$, $0 \leqslant j \leqslant k$,*

such that for all $0 \leqslant j < j' \leqslant k$, there is $1 \leqslant i \leqslant k$ such that for all $1 \leqslant i' \leqslant i$, we have $u_{i',j} = u_{i',j'}$, $v_{i',j} = v_{i',j'}$ and

a) either $|y_{i,j}| \neq |y_{i,j'}|$,

b) or $|y_{i,j}| = |y_{i,j'}| \neq 0$, the words $w_j x_{1,j} \dots x_{i,j}$ and $w_{j'} x_{1,j'} \dots x_{i,j'}$ have a mismatch, and the runs $\xrightarrow{w_j} q_{0,j} \xrightarrow{u_1 \dots u_i | x_{1,j} \dots x_{i,j}} q_{i,j}$ and $\xrightarrow{w_{j'}} q_{0,j'} \xrightarrow{u_1 \dots u_i | x_{1,j'} \dots x_{i,j'}} q_{i,j'}$ are $2M_{\mathcal{T}}.(|Q|^{k+1}+1)$-close.

Proof. The reverse implication is trivial, so we focus on the direct one. We consider a counter-example to the BTP_k and aim at deriving a counter example satisfying the above properties.

By Definition 4.4, there are

- states $q_{i,j}$ with $0 \leqslant i \leqslant k$, $0 \leqslant j \leqslant k$ and $q_{0,j}$ initial for all $0 \leqslant j \leqslant k$,
- words $u_{i,j}, v_{i,j} \in A^*$ with $1 \leqslant i \leqslant k$ and $0 \leqslant j \leqslant k$, and
- $k+1$ runs such that
  - $\xrightarrow{w_j} q_{0,j}$ for all $0 \leqslant j \leqslant k$, and
  - $q_{i-1,j} \xrightarrow{u_{i,j}|x_{i,j}} q_{i,j}$ and $q_{i,j} \xrightarrow{v_{i,j}|y_{i,j}} q_{i,j}$ for all $1 \leqslant i \leqslant k$, $0 \leqslant j \leqslant k$,

such that for all $0 \leqslant j < j' \leqslant k$, there is $1 \leqslant i \leqslant k$ such that for all $1 \leqslant i' \leqslant i$, we have $u_{i',j} = u_{i',j'}$, $v_{i',j} = v_{i',j'}$ and

$$\mathsf{delay}(w_j x_{1,j} \cdots x_{i,j}, w_{j'} x_{1,j'} \cdots x_{i,j'}) \neq \mathsf{delay}(w_j x_{1,j} \cdots x_{i,j} y_{i,j}, w_{j'} x_{1,j'} \cdots x_{i,j'} y_{i,j'}).$$

Let us consider a pair (j, j') of runs indices with $0 \leqslant j < j' \leqslant k$. Then there exists an index i (satisfying $i \leqslant \chi_{j,j'}$) such that the loop i induces a different delay. Let us take $i = \eta_{j,j'}$, which is the smallest such index, by definition.

First, if the delay difference is due to the length of the output words, *i.e.* $|y_{i,j}| \neq |y_{i,j'}|$, then we are done as we are in case a).

Second, there are two cases. If there is a mismatch in the words $w_j x_{1,j} \dots x_{i,j}$ and $w_{j'} x_{1,j'} \dots x_{i,j'}$, then we are in case b). Otherwise, we can assume that every loop of index at most i has output words of same length on components j and j', that is, for all $1 \leqslant i' \leqslant i$, $|y_{i',j}| = |y_{i',j'}|$. It means that there is a mismatch between the non-empty output words $y_{i,j}$ and $y_{i,j'}$. This loop can be unfolded to move the mismatch on output words of the runs leading to the loop. It remains to show that the runs are $2M_{\mathcal{T}}.(|Q|^{k+1}+1)$-close. If this is the case, then we are done and have proven that case b) is satisfied. Otherwise, we have that $|u_1 \cdots u_i| > |Q|^{k+1}$ and thus that there exists a synchronized loop (on the $k+1$ runs) whose output words on components j and j' have distinct length. But then we are back in case a). □

From Lemma 4.11, we can derive a non-deterministic decision procedure that will guess a counter example in three phases. The first phase will guess a skeleton of the counter-example. The second phase will materialise the loops of the counter-example and finally the third phase will materialise the backbone of the counter-example. We will show that the overall procedure executes in polynomial space.

Lemma 4.12. *The* BTP_k *problem is in* PSPACE *when* k *is given in unary.*

Proof. Let $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ be a finite-state transducer and k a natural.

Phase 1 We non-deterministically guess a skeleton of a counter-example. This skeleton consists of the following informations:

— $k+1$ vectors of states $\vec{q_0}, \dots, \vec{q_k}$ in Q^{k+1}, with components of $\vec{q_0}$ initial states,

— for all $0 \leqslant j < j' \leqslant k$, $\tau_{j,j'} \in \{a, b\}$ indicates, for the pair (j, j') of runs, whether case a) or case b) of Lemma 4.11 will be at fault.

— for all $0 \leqslant j < j' \leqslant k$, $\eta_{j,j'} \in \{1, \dots, k\}$ indicates the index of the loop that will exhibits said fault on the pair (j, j') of runs. Note that up to this index (included), the runs j and j' read the same input.

Phase 2 We non-deterministically materialise the loops of the counter-example. We build runs $\rho'_{i,j}$ looping around $\vec{q_i}[j]$, for all $1 \leqslant i \leqslant k$ and $0 \leqslant j \leqslant k$. For each pair (j, j'), we take care that

— if $i \leqslant \eta_{j,j'}$ then $\rho'_{i,j}$ and $\rho'_{i,j'}$ read the same input,

— if $i = \eta_{j,j'}$ then either $|\mathsf{out}(\rho'_{i,j})| \neq |\mathsf{out}(\rho'_{i,j'})|$ if $\tau_{j,j'} = a$, or $|\mathsf{out}(\rho'_{i,j})| = |\mathsf{out}(\rho'_{i,j'})| \neq 0$ if $\tau_{j,j'} = b$.

For each $1 \leqslant i \leqslant k$, we proceed to find cycles around $\vec{q_i}$ for all pairs (j, j') independently. Note that, in both the a) and b) cases, we can prove by contradiction that we can find such a cycle on an input word of length at most $2|Q|^{k+1}$. We let $L = 2M_{\mathcal{T}}|Q|^{k+1}$.

Let $1 \leqslant i \leqslant k$ and let $0 \leqslant j < j' \leqslant k$ such that $i = \eta_{j,j'}$. Using the same technique as in Section 3.3, we build a directed graph with vertices in $Q^{k+1} \times \{0, \ldots, L\}^2$. The two counters n_1, n_2 of the vertices allow to track the output of the distinguished runs j and j'. We add edges similarly to Section 3.3 based on the transitions of $\mathcal{T}$, taking care to read the same input, if necessary, and updating the two counters n_1, n_2, if necessary. We then test if we can reach from vertex $(\vec{q_i}, 0, 0)$ any vertex $(\vec{q_i}, n_1, n_2)$ such that either $n_1 \neq n_2$ if $\tau_{j,j'} = a$ or $n_1 = n_2 \neq 0$ if $\tau_{j,j'} = b$.

The size of the graph is in $O(|Q|^{k+1} \times \log_2(L))$. As reachability in a graph can be decided in non-deterministic logarithmic space, we obtain that finding such a cycle can also be done in PSPACE.

We then find a linear combination of these cycles to concatenate them and obtain a single cycle around $\vec{q_i}$ still respecting the constraints that the a) or b) cases require.

For each $1 \leqslant i \leqslant k$, we built $O(k^2)$ cycles, using polynomial space for each. Therefore, the overall process is in PSPACE.

Phase 3 We non-deterministically materialise the backbone of the counter-example. We can adapt the proof technique presented in Section 3.3 to decide the existence of a mismatch between two runs that stay $2M_{\mathcal{T}}.(|Q|^{k+1} + 1)$-close as follows. We let $N = 2M_{\mathcal{T}}.(|Q|^{k+1} + 1)$.

We build a graph which guesses the $k + 1$ runs in parallel, and according to the structure guessed in the skeleton (it indicates which runs share their inputs or not). For each pair of runs who should exhibit a mismatch, as indicated by the skeleton, we maintain a counter, which is bounded by N.

Additionally, for each pair of runs who should exhibit a mismatch, vertices allow to non-deterministically store the letter produced by the run which is ahead (ρ for instance), and then continue the simulation of ρ' until ρ' catches up ρ (*i.e.* the counter is equal to 0) and checks that the letter produced by ρ' is different from the one stored before. We then test if we can reach any vertex with all the counters equal to 0 and having different stored letters for each pair of runs who should exhibit a mismatch.

The size of the graph is $O(|Q|^{k+1} \times k^2 \log_2(N))$. Again, as reachability in a graph can be decided in non-deterministic logarithmic space, we obtain that finding such a mismatch can also be done in PSPACE.

Each of these three phases can be done in PSPACE, and we obtain an overall decision procedure in PSPACE. □

We now prove a lower bound for our decision procedure.

Lemma 4.13. *The* BTP_k *problem is* PSPACE*-hard when* k *is given in unary.*

Proof. We present a reduction of the emptiness of k deterministic finite state automata to the BTP_k problem.

Let $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be k deterministic finite-state automata over some alphabet A. Let $\#$ and $\$$ be two fresh symbols not in A. We build a functional FST $\mathcal{T}$ realising the function:

$$f : u\#\$^m\#\$^i \mapsto \$^{m\times i} \quad \text{for all } m \in \mathbb{N}, i \in \{1,\ldots,k\}, \text{ if } u \in \mathsf{dom}(\mathcal{D}_i)$$

We build $\mathcal{T}$ as the union of k sequential FSTs $\mathcal{T}_i$. We build each $\mathcal{T}_i$ from the corresponding $\mathcal{D}_i$ as follows :

— The states of $\mathcal{T}_i$ are the states of $\mathcal{D}_i$ and $i+1$ new states: $q_\#$ and q_j for all $j \in \{1,\ldots,i\}$.
— The initial function of $\mathcal{T}_i$ associates the initial state of $\mathcal{D}_i$ to ε.
— For each transition $p \xrightarrow{a} q$ of $\mathcal{D}_i$, we add a transition $p \xrightarrow{a|\varepsilon} q$ to $\mathcal{T}_i$.
— For each final state p of $\mathcal{D}_i$, we add a transition $p \xrightarrow{\#|\varepsilon} q_\#$.
— We add a self loop $q_\# \xrightarrow{\$|\$^i} q_\#$ and a transition $q_\# \xrightarrow{\#|\varepsilon} q_1$.
— For all $j \in \{1,\ldots,i-1\}$, we add a transition $q_j \xrightarrow{\#|\varepsilon} q_{j+1}$.
— The final function of $\mathcal{T}_i$ associates q_i to ε.

Observe that the size of $\mathcal{T}$ is polynomial in the sum of the sizes of the $\mathcal{D}_i$'s.

We now prove that $\mathcal{T}$ satisfies the BTP_{k-1} if and only if the intersection of the languages defined by the D_i's is empty.

Suppose that $\mathcal{T}$ does not satisfy the BTP_{k-1}. We thus have a counter example with k runs $\rho_1, \ldots, \rho_k$. For each pair of runs, there is a synchronized loop such that this loop is inducing a delay between these two runs and the two runs read the same input up to (and including) this loop. The delay must be induced by the self loops around the $q_\#$ states as those are the only productive transitions of the $\mathcal{T}_i$'s. As there is a delay, those two runs are in two different $\mathcal{T}_i$'s. Therefore, by the pigeon hole principle, every one of the k runs has a loop around the $q_\#$ state of a different $\mathcal{T}_i$. Consider the prefix $u\#$ of the input read by ρ_1. For all $i \in \{2,\ldots,k\}$, there is a synchronized loop such that this loop is inducing a delay between ρ_1 and ρ_i, and ρ_1 and ρ_i read the same input up to (and including) this loop. Therefore, every one of the k runs read the same word u before the first $\#$. This in turn means that all of the $\mathcal{D}_i$'s can read the word u, and thus the intersection of the languages defined by the D_i's is not empty.

Conversely, suppose now that the intersection of the languages defined by the D_i's is non-empty, and let u be a word in this intersection. We show that $[\![\mathcal{T}]\!]$ does not satisfy the Lipschitz property of order $k-1$. Fix a constant K, and consider the k input words $u_i = u\#\$^m\#\$^i$, for $i \in \{1,\ldots,k\}$ with $m = K.k$. For all $1 \leqslant j < j' \leqslant k$, we have $\mathsf{dist}_p(u_j, u_{j'}) < k$ and $\mathsf{dist}_p([\![\mathcal{T}]\!](u_j), [\![\mathcal{T}]\!](u_{j'})) \geqslant K.k$.

Therefore, $[\![\mathcal{T}]\!]$ does not satisfy the Lipschitz property of order $k-1$ and, by Theorem 4.3, $\mathcal{T}$ does not satisfy the BTP_{k-1}. □

Theorem 4.14. *The k-sequentiality problem is* PSPACE*-complete when k is given in unary.*

Proof. This follows from Lemmas 4.12 and 4.13. □

4.5 Minimisation of the Degree of Sequentiality

In this chapter, we have described two procedures: one to decide, given a functional finite-state transducer and a natural number k, whether this transducer admits a k-sequential equivalent, and one to build the k-sequential equivalent, under the proviso that it exists. In this subsection, we address the problem of finding the degree of sequentiality of a functional finite-state transducer, *i.e.* finding the minimal k such that this transducer can be realised by a k-sequential transducer.

The decision procedure for the k-sequentiality problem is in PSPACE. By carefully analysing the proof of Lemma 4.12, we can devise a deterministic algorithm to decide the problem in EXPTIME. Given a functional k-sequential finite-state transducer, we can proceed by dichotomy from k to determine its degree of sequentiality. We then need $log_2(k)$ calls to our decision procedure, giving an overall procedure still in EXPTIME.

Theorem 4.15. *Given a functional multi-sequential FST, we can compute its degree of sequentiality in* EXPTIME.

The problem of the register complexity of deterministic streaming string transducers was introduced by [AR13] and consists, given a deterministic streaming string transducer, in finding the minimal k such that there exists an equivalent deterministic streaming string transducer with k registers. We can apply the same strategy as above to solve this problem for the class of copyless appending deterministic streaming string transducers. Given a copyless appending deterministic streaming string transducer with k_0 register, we build an equivalent k_0-sequential (of polynomial size) and compute its degree of sequentiality k, by dichotomy from k_0. We can then build an equivalent copyless appending deterministic streaming string transducer with k register.

Corollary 4.16. *Given a copyless appending DSST, we can compute its register complexity in* EXPTIME.

Chapter 5

Sequentiality of String-to-Context Transducers

In this chapter, we present the work that we developed in [RV19] to characterise the functional string-to-context transducers that admit an equivalent sequential one. We wish to provide here a more complete picture of the problem

of the sequentiality of string-to-context transducers by defining an additional property similar to the bounded variation property of [Cho77] which was not in the initial publication.

In Chapter 2, we have seen that functional finite-state transducers are equivalent to copyful appending deterministic streaming string transducers. [DRT16] observed that the minimisation to one register of copyful appending deterministic streaming string transducers precisely coincides with the sequentiality problem of functional finite-state transducers. Thanks to the equivalence between functional string-to-context transducers and copyful concatenation-free deterministic streaming string transducers (*cf.* Chapter 2), we claim that solving the sequentiality problem of functional string-to-context transducers is a first important step towards solving the register minisation problem for copyful concatenation-free deterministic streaming string transducers.

In order to characterise the functional string-to-context transducers that admit an equivalent sequential one, we once more extend the results of [Cho77] around sequentiality. We generalise those for the string-to-context transducers by defining a contextual bounded variation property, a contextual Lipschitz property and a contextual twinning property. From these properties, we draw a construction for an equivalent sequential string-to-context transducer, if it exists, along with a decision procedure for the problem of the sequentiality of string-to-context transducers. The key tool behind this generalisation is a notion of distance that is appropriate to how the output is constructed by string-to-context transducers.

5.1 Preliminaries

5.1.1 Combinatorial Tools

Let A an alphabet. The *primitive period* of a word $x \in A^+$, denoted by $\mathsf{period}(x)$, is the shortest (unique) primitive word y such that $x \in y^+z$ for some z prefix of y.

Example 5.1. The primitive root and primitive period act differently. For instance, $\mathsf{root}(abcab) = abcab$ but $\mathsf{period}(abcab) = abc$.

The next lemma states a property of the primitive period that we will use throughout this chapter.

Lemma 5.1. *Let $v, x, z \in A^+$ and $w \in A^*$ such that $\mathsf{root}(v) \sim \mathsf{root}(x) \sim \mathsf{root}(z)$. If vwx is a factor of a word in z^* then $\mathsf{period}(vwx) \sim \mathsf{root}(z)$.*

Proof. Without loss of generality, consider z to be primitive. Let $u, y \in A^*$ and $i \geqslant 2$ such that $uvwxy = z^i$. There exist t_1, t_1', t_2, t_2' such that $z = t_1t_1' = t_2t_2'$ and $\mathsf{root}(v) = t_1't_1$ and $\mathsf{root}(x) = t_2't_2$. Then there exists $\alpha \leqslant 0$, $\beta \leqslant 0$ and

$\gamma > 0$ such that $u = t_1(t'_1t_1)^\alpha$, $y = (t'_2t_2)^\beta t'_2$, and $vwx = t'_1(t_1t'_1)^\gamma t_2$. Therefore $vwx \in (t'_1t_1)^+t'_1t_2$.

If $|t_2| \leqslant |t_1|$, then t'_1t_2 is a prefix of t'_1t_1. Otherwise, let $t \in A^+$ such that $t'_1t_2 = t'_1t_1t$, and then t is a prefix of t'_1t_1. In both cases, by definition, we have $\mathsf{period}(vwx) \sim z$. □

The next lemma recalls another classical result of combinatorics that we will use along with Fine and Wilf (Lemma 1.1).

Lemma 5.2 (Saarela, Theorem 4.3 of [Saa15]). *Let $m, n \geqslant 1$, $s_j, t_j \in A^*$ and $u_j, v_j \in A^+$. If $s_0u_1^is_1 \dots u_m^is_m = t_0v_1^it_1 \dots v_n^it_n$ holds for $m + n$ values of i, then it holds for all i.*

5.1.2 Factor Distance

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 $\mathsf{dist}_f(u, v)$, is defined as $\mathsf{dist}_f(u, v) = |u| + |v| - 2|\mathsf{lcf}(u, v)|$. 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 5.3. *dist_f is a distance.*

Proof. Let x, y, z words.

— **Symmetry:** It is trivial to prove that $\mathsf{dist}_f(x, y) = \mathsf{dist}_f(y, x)$.

— **Identity:** It is trivial to prove that $\mathsf{dist}_f(x, y) = 0 \iff x = y$.

— **Triangle Inequality:** We want to prove that $\mathsf{dist}_f(x, z) \leqslant \mathsf{dist}_f(x, y) + \mathsf{dist}_f(y, z)$. By definition, $\mathsf{dist}_f(x, y) = |x| + |y| - 2|\mathsf{lcf}(x, y)|$, $\mathsf{dist}_f(y, z) = |y| + |z| - 2|\mathsf{lcf}(y, z)|$ and $\mathsf{dist}_f(x, z) = |x| + |z| - 2|\mathsf{lcf}(x, z)|$. Let α a longest common factor of x and y, and β a longest common factor of y and z. Let $x_1, x_2, y_1, y_2, y_3, y_4, z_1, z_2$ words such that $x = x_1\alpha x_2$, $y = y_1\alpha y_2$, and $y = y_3\beta y_4$, $z = z_1\beta z_2$. Observe that $\mathsf{dist}_f(x, y) = |x_1| + |x_2| + |y_1| + |y_2|$ and $\mathsf{dist}_f(y, z) = |y_3| + |y_4| + |z_1| + |z_2|$. We observe six cases:

(i) $|y_1| \leqslant |y_3| < |y_1| + |\alpha|$ and $|y_4| \leqslant |y_2| < |y_4| + |\beta|$.
There exists γ such that $y_1\alpha = y_3\gamma$ and $\gamma y_2 = \beta y_4$. Then we have $|\alpha| \leqslant |y_3|+|\gamma|$ and $|\beta| \leqslant |y_2|+|\gamma|$. Yet, $|x| = |x_1|+|x_2|+|\alpha|$ and we obtain that $|x|-|\gamma| = |x_1|+|x_2|+|\alpha|-|\gamma| \leqslant |x_1|+|x_2|+|y_3|$. Also, $|z| = |z_1|+|z_2|+|\beta|$ and we obtain that $|z| - |\gamma| = |z_1| + |z_2| + |\beta| - |\gamma| \leqslant |z_1| + |z_2| + |y_2|$. Finally, we have that $|\mathsf{lcf}(x, z)| \geqslant |\gamma|$ as γ indeed is a common factor of

x and z. Then,

$$\begin{aligned}\mathsf{dist}_f(x,z) &= |x| + |z| - 2|\mathsf{lcf}(x,z)| \\ &\leqslant |x| + |z| - 2|\gamma| \\ &\leqslant |x_1| + |x_2| + |y_3| + |z_1| + |z_2| + |y_2| \\ &\leqslant \mathsf{dist}_f(x,y) + \mathsf{dist}_f(y,z)\end{aligned}$$

(ii) $|y_3| \leqslant |y_1| < |y_3| + |\alpha|$ and $|y_2| \leqslant |y_4| < |y_2| + |\beta|$.
This case is symmetrical to the previous case.

(iii) $|y_1| \leqslant |y_3| < |y_1| + |\alpha|$ and $|y_2| \leqslant |y_4| < |y_2| + |\beta|$.
Then we have $|\alpha| \leqslant |y_3| + |\beta| + |y_4|$. Yet, $|x| = |x_1| + |x_2| + |\alpha|$ and we obtain that $|x| - |\beta| = |x_1| + |x_2| + |\alpha| - |\beta| \leqslant |x_1| + |x_2| + |y_3| + |y_4|$. Also, $|z| = |z_1| + |z_2| + |\beta|$ and thus $|z| - |\beta| = |z_1| + |z_2|$. Finally, we have that $|\mathsf{lcf}(x,z)| \geqslant |\beta|$ as β indeed is a common factor of x and z. Then,

$$\begin{aligned}\mathsf{dist}_f(x,z) &= |x| + |z| - 2|\mathsf{lcf}(x,z)| \\ &\leqslant |x| + |z| - 2|\beta| \\ &\leqslant |x_1| + |x_2| + |y_3| + |y_4| + |z_1| + |z_2| \\ &\leqslant \mathsf{dist}_f(x,y) + \mathsf{dist}_f(y,z)\end{aligned}$$

(iv) $|y_3| \leqslant |y_1| < |y_3| + |\alpha|$ and $|y_4| \leqslant |y_2| < |y_4| + |\beta|$.
This case is symmetrical to the previous case.

(v) $|y_1| + |\alpha| \leqslant |y_3|$ and $|y_4| + |\beta| \leqslant |y_2|$.
Then, $|\alpha| \leqslant |y_3|$ and $|\beta| \leqslant |y_2|$. Yet, $|x| = |x_1| + |x_2| + |\alpha|$ and we obtain that $|x| \leqslant |x_1| + |x_2| + |y_3|$. Also, $|z| = |z_1| + |z_2| + |\beta|$ and we obtain that $|z| \leqslant |z_1| + |z_2| + |y_2|$. Then, as $|\mathsf{lcf}(x,z)| \geqslant 0$,

$$\begin{aligned}\mathsf{dist}_f(x,z) &= |x| + |z| - 2|\mathsf{lcf}(x,z)| \\ &\leqslant |x| + |z| \\ &\leqslant |x_1| + |x_2| + |y_3| + |z_1| + |z_2| + |y_2| \\ &\leqslant \mathsf{dist}_f(x,y) + \mathsf{dist}_f(y,z)\end{aligned}$$

(vi) $|y_3| + |\alpha| \leqslant |y_1|$ and $|y_2| + |\beta| \leqslant |y_4|$.
This case is symmetrical to the previous case. □

We now prove some classical properties of distances.

Lemma 5.4. *For all $c, c' \in \mathcal{C}(B)$ and $w \in B^*$, we have $\mathsf{dist}_f(c[w], c'[w]) \leqslant |c| + |c'|$.*

Proof. It is easy to see that $|\mathsf{lcf}(c[w], c'[w])| \geqslant |w|$. Then, we have:

$$\begin{aligned}\mathsf{dist}_f(c[w], c'[w]) &= |c| + |c'| + |w| + |w| - 2|\mathsf{lcf}(c[w], c'[w])| \\ &\leqslant |c| + |c'| + |w| + |w| - 2|w| \\ &\leqslant |c| + |c'|\end{aligned}$$ □

Lemma 5.5. *For all $c, c' \in \mathcal{C}(B)$ and $w, w' \in B^*$, we have*

$$\mathit{dist}_f(w, w') - |c| - |c'| \leqslant \mathit{dist}_f(c[w], c'[w']) \leqslant \mathit{dist}_f(w, w') + |c| + |c'|.$$

Proof. Using the triangle inequality and Lemma 5.4, we have:

$$\begin{aligned}\mathsf{dist}_f(w, w') &\leqslant \mathsf{dist}_f(w, c[w]) + \mathsf{dist}_f(c[w], c'[w']) + \mathsf{dist}_f(c'[w'], w') \\ &\leqslant |c| + \mathsf{dist}_f(c[w], c'[w']) + |c'|\end{aligned}$$

Again, using the triangle inequality and Lemma 5.4, we have:

$$\begin{aligned}\mathsf{dist}_f(c[w], c'[w']) &\leqslant \mathsf{dist}_f(c[w], w) + \mathsf{dist}_f(w, w') + \mathsf{dist}_f(w', c'[w']) \\ &\leqslant |c| + \mathsf{dist}_f(w, w') + |c'|\end{aligned}$$ □

Let us also prove a simple property of lcf.

Lemma 5.6. *For all $u_1, u_2, v_1, v_2 \in B^*$, $|\mathit{lcf}(u_1u_2, v_1v_2)| \geqslant |\mathit{lcs}(u_1, v_1)| + |\mathit{lcp}(u_2, v_2)|$.*

Proof. Let $x = \mathsf{lcs}(u_1, u_2)$ and $y = \mathsf{lcp}(u_2, v_2)$. Then there exist $u_1', u_2', v_1', v_2' \in B^*$ such that $u_1 = u_1'x$, $v_1 = v_1'x$, $u_2 = yu_2'$ and $v_2 = yv_2'$. Then we have $u_1u_2 = u_1'xyu_2'$ and $v_1v_2 = v_1'xyv_2'$. Therefore xy is a factor of u_1u_2 and v_1v_2 and, by definition of the longest common factor, $|\mathsf{lcp}(u_1u_2, v_1v_2) \geqslant |xy| \geqslant |\mathsf{lcs}(u_1, v_1)| + |\mathsf{lcp}(u_2, v_2)|$. □

5.2 Characterisation of Sequential S2Cs

We first present the adaptation of the bounded variation, Lipschitz and twinning properties to string-to-context transducers.

5.2.1 Contextual Bounded Variation

We adapt the classical bounded variation property of [Cho77] by using the factor distance to denote how close the output words must remain. Notice that we still need the prefix distance to set bounds for the input words.

Definition 5.1 (Contextual Bounded Variation Property (CBV))**.** Let A, B be two alphabets. A function f from A^* to B^* satisfies the *contextual bounded variation property* if for all $m \in \mathbb{N}$, there exists $M \in \mathbb{N}$ such that for all $u, v \in \mathsf{dom}(f)$, if $\mathsf{dist}_p(u, v) \leqslant m$ then $\mathsf{dist}_f(f(u), f(v)) \leqslant M$.

5.2.2 Contextual Lipschitz Property

The factor distance is also appropriate to generalise the Lipschitz property.

Definition 5.2 (Contextual Lipschitz Property (CLip))**.** Let A, B be two alphabets. A function f from A^* to B^* satisfies the *contextual Lipschitz property* if there exists $K \in \mathbb{N}$ such that $\forall u, v \in \mathsf{dom}(f), \mathsf{dist}_f(f(u), f(v)) \leqslant K\, \mathsf{dist}_p(u, v)$.

Example 5.2. The function f_{mirror} defined in Example 1.9 obviously satisfies the contextual Lipschitz property with coefficient 1. Indeed, let $u, v \in \{\mathsf{a}, \mathsf{b}\}^*$. We have $f(u) = \tilde{u}$ and $f(v) = \tilde{v}$. By Lemma 5.6, $|\mathsf{lcf}(\tilde{u}, \tilde{v})| \geqslant |\mathsf{lcs}(\tilde{u}, \tilde{v})| = |\mathsf{lcp}(u, v)|$. Then, $\mathsf{dist}_f(f(u), f(v)) = |\tilde{u}| + |\tilde{v}| - 2|\mathsf{lcf}(\tilde{u}, \tilde{v})| \leqslant |u| + |v| - 2|\mathsf{lcp}(u, v)| = \mathsf{dist}_p(u, v)$.

Similarly, the function $f_{mirror \cdot id}$ defined in Example 2.1 satisfies the contextual Lipschitz property with coefficient 2. Indeed, let $u, v \in \{\mathsf{a}, \mathsf{b}\}^*$. We have $f(u) = \tilde{u}u$ and $f(v) = \tilde{v}v$. By Lemma 5.6, $|\mathsf{lcf}(\tilde{u}u, \tilde{v}v)| \geqslant |\mathsf{lcs}(\tilde{u}, \tilde{v})| + |\mathsf{lcp}(u, v)| = 2|\mathsf{lcp}(u, v)|$. Then, $\mathsf{dist}_f(f(u), f(v)) = |\tilde{u}u| + |\tilde{v}v| - 2|\mathsf{lcf}(\tilde{u}u, \tilde{v}v)| \leqslant 2|u| + 2|v| - 4|\mathsf{lcp}(u, v)| = 2\mathsf{dist}_p(u, v)$.

Example 5.3. The function $f_{mirror \cdot last} : u \in \{\mathsf{a}, \mathsf{b}\}^+ \mapsto \tilde{u} \cdot \mathsf{last}(u)^{|u|}$ does not satisfy the contextual Lipschitz property. Indeed let $K \in \mathbb{N}$ and take $u = \mathsf{a}^K\mathsf{a}$, $v = \mathsf{a}^K\mathsf{b}$. We have $f(u) = \mathsf{a}^{2K+2}$ and $f(v) = \mathsf{ba}^K\mathsf{b}^{K+1}$. Then, $\mathsf{dist}_f(f(u), f(v)) = 2(K+2) > K.\mathsf{dist}_p(u, v) = 2K$.

In the next lemma, we state that the contextual Lipschitz property implies the contextual bounded variation property. We will later prove that they actually are equivalent.

Lemma 5.7. *Let A, B be two alphabets. If a function f from A^* to B^* satisfies the contextual Lipschitz property then it satisfies the contextual bounded variation property.*

Proof. Let f that satisfies the contextual Lipschitz property and let $K \in \mathbb{N}$ such that for all $u, v \in \mathsf{dom}(f)$, $\mathsf{dist}_f(f(u), f(v)) \leqslant K.\mathsf{dist}_p(u, v)$. We prove that f satisfies the contextual bounded variation property. Let $m \in \mathbb{N}$ and define $M = Km$. If $u, v \in \mathsf{dom}(f)$ and $\mathsf{dist}_p(u, v) \leqslant m$ then we have $\mathsf{dist}_f(f(u), f(v)) \leqslant K.\mathsf{dist}_p(u, v) \leqslant M$. □

5.2.3 Contextual Twinning Property

Again, we can extend the twinning property thanks to the factor distance.

Definition 5.3 (Contextual Twinning Property (CTP))**.** 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 states, we have for all $j \geqslant 0$, $\mathsf{dist}_f(e_1^j d_1 c_1[\varepsilon], e_2^j d_2 c_2[\varepsilon]) \leqslant L$. An S2C satisfies the *contextual twinning property* if there exists $L \in \mathbb{N}$ such that any two of its states are L-contextually twinned.

Example 5.4. Figure 5.1 depicts $\mathcal{T}_{mirror \cdot last}$, computing the function $f_{mirror \cdot last}$ defined in Example 5.3. $\mathcal{T}_{mirror \cdot last}$ does not satisfy the contextual twinning property. Indeed, in search of a contradiction, assume that $\mathcal{T}_{mirror \cdot last}$ does satisfy the twinning property and let $L \in \mathbb{N}$ such that any two states of $\mathcal{T}_{mirror \cdot last}$ are L-twinned. Now, consider two loops around q_a and q_b: $p_1 = q_1 = q_a$, $p_2 = q_2 = q_b$, $u = \varepsilon$ and $v = \mathsf{a}$. Then we have $c_1 = c_2 = d_1 = d_2 = (\varepsilon, \varepsilon)$, $e_1 = (\mathsf{a}, \mathsf{a})$ and $e_2 = (\mathsf{a}, \mathsf{b})$. Thus $\mathrm{dist}_f(e_1^L d_1 c_1[\varepsilon], e_2^L d_2 c_2[\varepsilon]) = 2L > L$ and we have a contradiction.

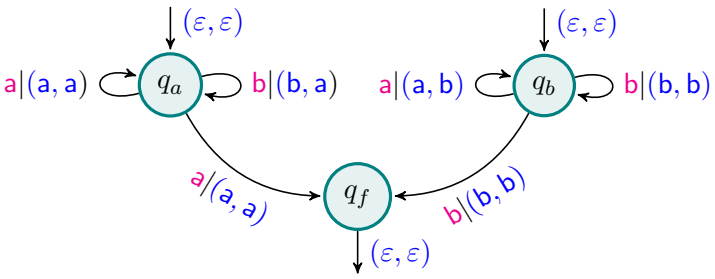


Figure 5.1 – The S2C $\mathcal{T}_{mirror \cdot last}$.

5.2.4 Sequentialisation Theorem for S2Cs

Our main result is the following theorem, which characterises the functional string-to-context transducers admitting an equivalent sequential string-to-context transducer.

Theorem 5.8. *Let A, B be two alphabets. Let $\mathcal{T}$ be a functional S2C from A^* to B^*. The following assertions are equivalent:*

1. *$[\![\mathcal{T}]\!]$ satisfies the contextual Lipschitz property,*
2. *$[\![\mathcal{T}]\!]$ satisfies the contextual bounded variation property,*
3. *$\mathcal{T}$ satisfies the contextual twinning property,*
4. *$[\![\mathcal{T}]\!]$ can be realised by a sequential S2C.*

Proof. The implication from 1 to 2 was proved in Lemma 5.7. The implications from 4 to 1 and from 2 to 3 are proved in Propositions 5.9 and 5.10. The implication from 3 to 4 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 5.33 and used in Section 5.4 to describe the construction of an equivalent sequential S2C. □

Proposition 5.9. *Let A, B be two alphabets. Let $\mathcal{T}$ be a sequential S2C realizing the function f from A^* to B^*. Then f satisfies the contextual Lipschitz property.*

Proof. We will prove that f satisfies the Lipschitz property 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 $u = w.u'$ and $v = w.v'$, with $0 \leqslant |u'|$ and $0 \leqslant |v'|$. As $\mathcal{T}$ is sequential, we have two runs in $\mathcal{T}$

$$\xrightarrow{c_1} p \xrightarrow{w|c_2} q \xrightarrow{u'|d_1} r \xrightarrow{d_2} \quad \text{and} \quad \xrightarrow{c_1} p \xrightarrow{w|c_2} q \xrightarrow{v'|e_1} s \xrightarrow{e_2}$$

such that $[\![\mathcal{T}]\!](u) = d_2d_1c_2c_1[\varepsilon]$ and $[\![\mathcal{T}]\!](v) = e_2e_1c_2c_1[\varepsilon]$. We also have $|d_1| \leqslant M_{\mathcal{T}}\,|u'|$, $|e_1| \leqslant M_{\mathcal{T}}\,|v'|$, $|d_2| \leqslant M_{\mathcal{T}}$, and $|e_2| \leqslant M_{\mathcal{T}}$. Finally, as $u \neq v$, we have $\mathsf{dist}_p(u, v) = |u'| + |v'| \geqslant 1$ and we obtain:

$$\begin{aligned}
\mathsf{dist}_f(f(u), f(v)) &\leqslant |d_2d_1| + |e_2e_1| \\
&\leqslant M_{\mathcal{T}}\,(2 + |u'| + |v'|) \\
&\leqslant 3M_{\mathcal{T}}\,(|u'| + |v'|) \\
&\leqslant 3M_{\mathcal{T}}\,\mathsf{dist}_p(u, v)
\end{aligned}$$

□

Proposition 5.10. *Let A, B be two alphabets. Let $\mathcal{T}$ be a functional S2C realizing the function f from A^* to B^*. If f satisfies the contextual bounded variation property, then $\mathcal{T}$ satisfies the contextual twinning property.*

Proof. We denote by n the number of states of $\mathcal{T}$. Suppose that f satisfies the contextual bounded variation property, and let $N \in \mathbb{N}$ such that for all $u, v \in \mathsf{dom}(f)$, if $\mathsf{dist}_p(u, v) \leqslant 2n$ then $\mathsf{dist}_f(f(u), f(v)) \leqslant N$. We consider an instance of the contextual twinning property in $\mathcal{T}$:

$$\xrightarrow{c_1} p_1 \xrightarrow{u|d_1} q_1 \xrightarrow{v|e_1} q_1 \quad \text{and} \quad \xrightarrow{c_2} p_2 \xrightarrow{u|d_2} q_2 \xrightarrow{v|e_2} q_2$$

As $\mathcal{T}$ is trimmed, there exist runs

$$q_1 \xrightarrow{w_1|f_1} r_1 \xrightarrow{g_1} \quad \text{and} \quad q_2 \xrightarrow{w_2|f_2} r_2 \xrightarrow{g_2}$$

with $|w_1| \leqslant n$ and $|w_2| \leqslant n$. We consider the input words $\alpha_j = uv^jw_1$ and $\beta_j = uv^jw_2$, for all $j \geqslant 0$. We have, for all $j \geqslant 0$, $\mathsf{dist}_p(\alpha_j, \beta_j) \leqslant |w_1| + |w_2| \leqslant 2n$. Therefore, for all $j \geqslant 0$, $\mathsf{dist}_f(f(\alpha_j), f(\beta_j)) \leqslant N$.

By using the triangle inequality twice, we obtain that, for all $j \geqslant 0$:

$$\begin{aligned}
\mathsf{dist}_f(e_1^jd_1c_1[\varepsilon], e_2^jd_2c_2[\varepsilon]) &\leqslant \mathsf{dist}_f(g_1f_1e_1^jd_1c_1[\varepsilon], g_2f_2e_2^jd_2c_2[\varepsilon]) + 2(n+1)M_{\mathcal{T}} \\
&\leqslant \mathsf{dist}_f(f(\alpha_j), f(\beta_j)) + 2(n+1)M_{\mathcal{T}} \\
&\leqslant N + 2(n+1)M_{\mathcal{T}}
\end{aligned}$$

□

5.3 Combinatorial Analysis

Recall from Chapter 3, that 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 $\mathsf{root}(y_1) \sim \mathsf{root}(y_2)$.

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 $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ that satisfies the contextual twinning property.

5.3.1 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 5.5. Figure 5.2a 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 5.2c. Figure 5.2b 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 5.2d.

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

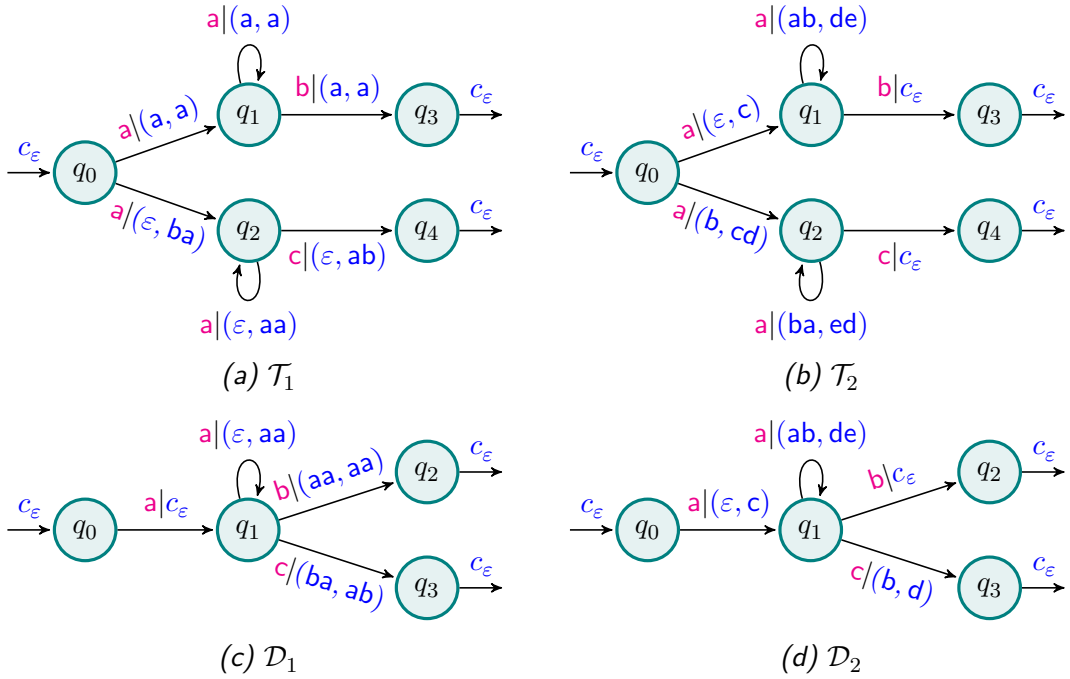
Definition 5.4 (Lasso, Aligned/Commuting/Non-commuting lasso)**.** A *lasso* around a state q is a run ρ of the form $\xrightarrow{c} p \xrightarrow{u|d} q \xrightarrow{v|e} q$ with p an initial state. ρ is said to be *productive*, if $|e| \neq 0$. We say that ρ is:

— *aligned* w.r.t. f and w, for some $f \in \mathcal{C}(B)$ and $w \in B^*$, denoted by $(f, w)-\mathrm{aligned}$, 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 by $x-\mathrm{commuting}$, if there exists a context $f \in \mathcal{C}(B)$ such that for all $i > 0$, 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 *weakly balanced* if $|e_1| = |e_2|$ and *strongly balanced* if $\|e_1\| = \|e_2\|$.



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

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

Figure 5.2 – 5.2a An S2C $\mathcal{T}_1$ computing the function that maps $a^n b$ to a^{2n+2} and $a^n c$ to $ba^{2n}b$. 5.2c A sequential S2C $\mathcal{D}_1$ equivalent to $\mathcal{T}_1$. 5.2b 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 $b(ab)^{n-1}c(de)^{n-1}d$. 5.2d A sequential S2C $\mathcal{D}_2$ equivalent to $\mathcal{T}_2$.

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_1H_2, where H_1 starts in initial states and 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 5.5 is formalised in the following lemma:

Lemma 5.11. *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]

1. Because we only consider lassos around pairwise distinct states, both $\mathsf{split}_c(x, H_1, H_2)$ and

— *or there exist* $f \in \mathcal{C}(B)$ *and* $w \in B^*$ *such that* ρ_j *is non-commuting and* $(f,w)-\text{aligned}$ *for all* $j \in \{1,\dots,k\}$. *In this case, we say that the lasso* H_1H_2 *is* $(f,w)-\text{aligned}$, *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]

Example 5.6. We consider the example S2C in Figure 5.2. The lasso in $\mathcal{T}_1^2$ around (q_1,q_2) is $\mathsf{a}-\text{commuting}$. 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 $((\mathsf{ab},\mathsf{de}),\mathsf{c})-\text{aligned}$. We can compute $\{(q_1,c_\varepsilon),(q_2,(\mathsf{b},\mathsf{d}))\}$ as a possible split_{nc}.

In the remainder of this subsection, we will prove Lemma 5.11. We first need to study the combinatorial properties of two synchronised lassos. We will later lift those to k synchronised lassos.

5.3.1.1 Properties of Two Synchronised Lassos

Lemma 5.12. *For any two synchronised lassos* ρ_1 *and* ρ_2, *we have that*
— *either* ρ_1 *and* ρ_2 *are non-productive*
— *or* ρ_1 *and* ρ_2 *are productive and weakly-balanced, and there exists* $x \in B^+$ *primitive such that* ρ_1 *and* ρ_2 *are* $x-\text{commuting}$,
— *or* ρ_1 *and* ρ_2 *are productive, strongly-balanced and non-commuting, and there exists* $f \in \mathcal{C}(B)$ *and* $w \in B^*$ *such that* ρ_1 *and* ρ_2 *are* $(f,w)-\text{aligned}$.

In order to prove Lemma 5.12, we first need some preliminary combinatorial results.

Lemma 5.13. *Let* $c_1,c_2,d_1,d_2 \in \mathcal{C}(B)$. *If for all* $i \in \mathbb{N}$, $\mathsf{dist}_f(d_1^i c_1[\varepsilon], d_2^i c_2[\varepsilon]) \leqslant L$, *then there exist* $e_1,e_2 \in \mathcal{C}(B)$ *such that there exist infinitely many integers* $i \in \mathbb{N}$, *such that* $e_1^{-1}d_1^i c_1[\varepsilon] = e_2^{-1}d_2^i c_2[\varepsilon]$

Proof. Suppose that for all $i \in \mathbb{N}$, $\mathsf{dist}_f(d_1^i c_1[\varepsilon], d_2^i c_2[\varepsilon]) \leqslant L$. Then for all $i \in \mathbb{N}$, there exist $f_1,f_2 \in \mathcal{C}(B)$, such that $f_1^{-1}d_1^i c_1[\varepsilon] = f_2^{-1}d_2^i c_2[\varepsilon]$ and $|f_1|+|f_2| \leqslant L$. Let $C_L = \{(f_1,f_2) \mid |f_1|+|f_2| \leqslant L\}$. C_L is finite. Thus there exists some $(e_1,e_2) \in C_L$ such that there exist infinitely many integers $i \in \mathbb{N}$ such that $e_1^{-1}d_1^i c_1[\varepsilon] = e_2^{-1}d_2^i c_2[\varepsilon]$. □

Lemma 5.14. *Let* $u_1,w_1,u_2,w_2 \in B^*$ *and* $v_1,v_2 \in B^+$ *such that* $|v_1| = |v_2|$ *and* $u_1v_1^iw_1 = u_2v_2^iw_2$ *for all* $i \in \mathbb{N}$. *Then there exists* $x \in B^+$ *and* $f_1,f_2 \in \mathcal{C}(B)$ *such that for all* $i \geqslant 1$, *there exist* $k \in \mathbb{N}$ *such that* $v_1^i = f_1[x^k]$ *and* $v_2^i = f_2[x^k]$.

Proof. There exists i_0 sufficiently large so that $v_1^{i_0}$ and $v_2^{i_0}$ overlap with a common factor of length greater than $|v_1|+|v_2|-\mathsf{gcd}(|v_1|,|v_2|)$. Thus, by Lemma 1.1, $\mathsf{root}(v_1) \sim \mathsf{root}(v_2)$.

[1] $\mathsf{split}_{nc}(f,w,H_1,H_2)$ are partial functions from Q to $\mathcal{C}(B)$.

Let $t, t' \in B^*$ and $\alpha, \beta \geqslant 1$ such that $v_1 = (tt')^\alpha$ and $v_2 = (t't)^\beta$. We choose $x = \mathsf{root}(v_1) = tt'$, $f_1 = (tt', \varepsilon)$ and $f_2 = (t', t)$. Then for all $i \geqslant 1$, let $k = \alpha i - 1 \geqslant 0$, and we have $v_1^i = (tt')^{\alpha i} = tt'(tt')^{\alpha i - 1} = f_1[x^k]$ and $v_2^i = (t't)^{\alpha i} = t'x^{\alpha i - 1}t = f_2[x^k]$. □

Lemma 5.15. *Let $u_1, w_1, u_2, w_2, y_2 \in B^*$ and $v_1, v_2, x_2 \in B^+$ such that $|v_1| = |v_2| + |x_2|$ and $u_1 v_1^i w_1 = u_2 v_2^i w_2 x_2^i y_2$ for all $i \in \mathbb{N}$. Then there exists $x \in B^+$ and $f_1, f_2 \in \mathcal{C}(B)$ such that for all $i \geqslant 1$, there exist $k \in \mathbb{N}$ such that $v_1^i = f_1[x^k]$ and $v_2^i w_2 x_2^i = f_2[x^k]$.*

Proof. There exists m_0 sufficiently large so that $v_1^{m_0}$ overlap with both $v_2^{m_0}$ and $x_2^{m_0}$ with common factors of length greater than $|v_1| + |v_2| - \mathsf{gcd}(|v_1|, |v_2|)$ and $|v_1| + |x_2| - \mathsf{gcd}(|v_1|, |x_2|)$. Thus, by Lemma 1.1, $\mathsf{root}(v_1) \sim \mathsf{root}(v_2)$ and $\mathsf{root}(v_1) \sim \mathsf{root}(x_2)$. As $|v_1| = |v_2| + |x_2|$, we have that $|v_2 w_2 x_2| \geqslant |v_1|$. Yet $v_2 w_2 x_2$ is a factor of v_1^*, then, by Lemma 5.1, $\mathsf{period}(v_2 w_2 x_2) \sim \mathsf{root}(v_1)$.

Let $t_1, t_1', t_2, t_2' \in B^*$ and $\alpha, \beta, \gamma \geqslant 1$ such that $t_1 t_1' = t_2 t_2'$, and $v_1 = (t_1 t_1')^\alpha$, $v_2 = (t_1' t_1)^\beta$ and $x_2 = (t_2' t_2)^\gamma$. Note that $\alpha = \beta + \gamma$. Also we have $v_2 w_2 x_2 = t_1'(t_1 t_1')^\theta t_2$, for some $\theta \geqslant 0$. We choose $x = \mathsf{root}(v_1) = t_1 t_1'$, $f_1 = ((t_1 t_1')^\alpha, \varepsilon)$ and $f_2 = (t_1'(t_1 t_1')^\theta, t_2)$. Then for all $i \geqslant 1$, let $k = \alpha(i-1) \geqslant 0$, and we have $v_1^i = (t_1 t_1')^{\alpha i} = (t_1 t_1')^\alpha x^{\alpha(i-1)} = f_1[x^k]$ and $v_2^i w_2 x_2^i = v_2^{i-1} v_2 w_2 x_2 x_2^{i-1} = (t_1' t_1)^{\beta(i-1)} t_1'(t_1 t_1')^\theta t_2 (t_2' t_2)^{\gamma(i-1)} = t_1'(t_1 t_1')^\theta x^{\alpha(i-1)} t_2 = f_2[x^k]$. □

Lemma 5.16. *Let $u_1, w_1, y_1, u_2, w_2, y_2 \in B^*$ and $v_1, x_1, v_2, x_2 \in B^+$ such that $|v_1| + |x_1| = |v_2| + |x_2|$ and $u_1 v_1^i w_1 x_1^i y_1 = u_2 v_2^i w_2 x_2^i y_2$ for all $i \in \mathbb{N}$. Then we have*

- *either $|v_1| \neq |v_2|$ and $|x_1| \neq |x_2|$, and there exists $x \in B^+$ and $f_1, f_2 \in \mathcal{C}(B)$ such that for all $i \geqslant 1$, there exist $k \in \mathbb{N}$ such that $v_1^i w_1 x_1^i = f_1[x^k]$ and $v_2^i w_2 x_2^i = f_2[x^k]$.*
- *or $|v_1| = |v_2|$ and $|x_1| = |x_2|$, and there exist $w \in B^*$ and $f, g_1, g_2 \in \mathcal{C}(B)$ such that for all $i \in \mathbb{N}$, $v_1^i w_1 x_1^i = g_1 f^i[w]$ and $v_2^i w_2 x_2^i = g_2 f^i[w]$.*

Proof. If $|v_1| \neq |v_2|$ and $|x_1| \neq |x_2|$, suppose $|v_1| > |v_2|$. There exists i_0 sufficiently large so that $v_1^{i_0}$ overlap with both $x_2^{i_0}$ with a common factor of length greater than $|v_1| + |x_2| - \mathsf{gcd}(|v_1|, |x_2|)$. Thus, by Lemma 1.1, $\mathsf{root}(v_1) \sim \mathsf{root}(x_2)$. Using the same argument, we have that $\mathsf{root}(v_1) \sim \mathsf{root}(v_2)$ and $\mathsf{root}(x_1) \sim \mathsf{root}(x_2)$. As $|v_1| + |x_1| = |v_2| + |x_2|$, we have that $|v_2| + |x_2| \geqslant |v_1|$ and thus $|v_2 w_2 x_2| \geqslant |v_1|$. Yet $v_2 w_2 x_2$ is a factor of v_1^*, then, by Lemma 5.1, $\mathsf{period}(v_2 w_2 x_2) \sim \mathsf{root}(v_1)$. Symmetrically, $\mathsf{period}(v_1 w_1 x_1) \sim \mathsf{root}(x_2)$.

Let $t_1, t_1', t_2, t_2', t_3, t_3' \in B^*$ and $\alpha, \beta, \gamma, \delta \geqslant 1$ such that $t_1 t_1' = t_2 t_2' = t_3 t_3'$, and $v_1 = (t_1 t_1')^\alpha$, $x_1 = (t_2' t_2)^\beta$, $v_2 = (t_1' t_1)^\gamma$ and $x_2 = (t_3' t_3)^\delta$. Note that $\alpha + \beta = \gamma + \delta$. Also we have $v_1 w_1 x_1 = (t_1 t_1')^{\theta_1} t_2$ and $v_2 w_2 x_2 = t_1'(t_1 t_1')^{\theta_2} t_3$, for some $\theta_1, \theta_2 \geqslant 0$. We choose $x = \mathsf{root}(v_1) = t_1 t_1'$ $f_1 = ((t_1 t_1')^{\theta_1}, t_2)$ and $f_2 = (t_1'(t_1 t_1')^{\theta_2}, t_3)$. Then for all $i \geqslant 1$, let $k = (\alpha + \beta)(i-1) \geqslant 0$, and we have $v_1^i w_1 x_1^i = v_1^{i-1} v_1 w_1 x_1 x_1^{i-1} = (t_1 t_1')^{\theta_1}(t_1 t_1')^{(\alpha+\beta)(i-1)} t_2 = f_1[x^k]$ and $v_2^i w_2 x_2^i = v_2^{i-1} v_2 w_2 x_2 x_2^{i-1} = t_1'(t_1 t_1')^{\theta_2}(t_1 t_1')^{(\gamma+\delta)(i-1)} t_3 = f_2[x^k]$.

If $|v_1| < |v_2|$, we obtain the same result.

If $|v_1| = |v_2|$ and $|x_1| = |x_2|$, we only have that $\mathsf{root}(v_1) \sim \mathsf{root}(v_2)$ and $\mathsf{root}(x_1) \sim \mathsf{root}(x_2)$. If $|u_1| < |u_2|$, let v such that we have $u_2 = u_1 v$ and $v_1 v = v v_2$; if $|u_1| = |u_2|$, let $v = \varepsilon$ and we have $u_1 = u_2$ and $v_1 = v_2$; if $|u_1| > |u_2|$, let v such that we have $u_1 = u_2 v$ and $v_2 v = v v_1$. Similarly, if $|y_1| < |y_2|$, let x such that we have $y_2 = x y_1$ and $x_2 x = x x_1$; if $|y_1| = |y_2|$, let $x = \varepsilon$ and we have $y_1 = y_2$ and $x_1 = x_2$; if $|y_1| > |y_2|$, let x such that we have $y_1 = x y_2$ and $x_1 x = x x_2$.

Finally, from v and y, we obtain that

— $v_1 v = v v_2$, $w_1 = v w_2 x$, $x_2 x = x x_1$, or
— $v_1 v = v v_2$, $w_1 x = v w_2$, $x_1 x = x x_2$, or
— $v_2 v = v v_1$, $v w_1 = w_2 x$, $x_2 x = x x_1$, or
— $v_2 v = v v_1$, $v w_1 x = w_2$, $x_1 x = x x_2$.

We handle the first case. The others are similar. We choose $f = (v_2, x_2)$, $w = w_2$, and $g_1 = (v, x)$ and $g_2 = (\varepsilon, \varepsilon)$. Then for all $i \in \mathbb{N}$, $v_1^i w_1 x_1^i = v_1^i w_1 x_1^i = v_1^i v w_2 x x_1^i = v v_2^i w_2 x_2^i x = v(f^i[w])x = g_1 f^i[w]$ and $v_2^i w_2 x_2^i = g_2 f^i[w]$. □

Lemma 5.17. *Let $f \in \mathcal{C}(B)$, $w \in B^*$, and $x \in B^+$ a primitive word. Let ρ_1 and ρ_2 be two synchronised, productive, strongly-balanced and $(f,w)-$aligned lassos. If ρ_1 is $x-$commuting, then ρ_2 is $x-$commuting.*

Proof. Let $\rho_1 : \xrightarrow{c_1} p_1 \xrightarrow{u_1|d_1} q_1 \xrightarrow{u_2|e_1} q_1$ and $\rho_2 : \xrightarrow{c_2} p_2 \xrightarrow{u_1|d_2} q_2 \xrightarrow{u_2|e_2} q_2$.

We have that $\|e_1\| = \|e_2\|$, and ρ_1 and ρ_2 are $(f,w)-$aligned. By Definition 5.4, there exist some contexts $g_1, g_2 \in \mathcal{C}(B)$ such that for all $i \geqslant 0$, $e_1^i d_1 c_1[\varepsilon] = g_1 f^i[w]$ and $e_2^i d_2 c_2[\varepsilon] = g_2 f^i[w]$. If ρ_1 is $x-$commuting then, by Definition 5.4, there exist $h \in \mathcal{C}(B)$ such that for all $i \geqslant 0$, there exists $j \geqslant 0$ such that $e_1^i d_1 c_1[\varepsilon] = h[x^j]$. Hence, there exists $k \geqslant 0$ and $h' \in \mathcal{C}(B)$ such that $f^i w = h'[x^k]$ and then $g_2 f^i[w] = g_2 h'[x^k]$. Therefore, by Definition 5.4, ρ_2 is $x-$commuting. □

We can now prove Lemma 5.12.

Proof of Lemma 5.12. Let $\rho_1 : \xrightarrow{c_1} p_1 \xrightarrow{u_1|d_1} q_1 \xrightarrow{u_2|e_1} q_1$ and $\rho_2 : \xrightarrow{c_2} p_2 \xrightarrow{u_1|d_2} q_2 \xrightarrow{u_2|e_2} q_2$. By Lemma 5.13, there exist $f_1, f_2 \in \mathcal{C}(B)$ such that there exist infinitely many integers $i \in \mathbb{N}$ such that $f_1^{-1} e_1^i d_1 c_1[\varepsilon] = f_2^{-1} e_2^i d_2 c_2[\varepsilon]$. Then we have that $|e_1| = |e_2|$. We observe 10 cases.

If $|e_1| = 0$ or $|e_2| = 0$ then $|e_1| = |e_2| = 0$ and ρ_1, ρ_2 are not productive.

If $e_1, e_2 \in B^+ \times \{\varepsilon\}$, then there exist $g_1, g_2 \in \mathcal{C}(B)$ and i_0 such that there exist infinitely many integers $i \geqslant i_0$, such that $g_1[\overleftarrow{e_1}^{i-i_0}] = g_2[\overleftarrow{e_2}^{i-i_0}]$. Therefore, by Lemma 5.2, we obtain that for all $i \in \mathbb{N}$, $g_1[\overleftarrow{e_1}^{i}] = g_2[\overleftarrow{e_2}^{i}]$. Then by Lemma 5.14, there exists $x \in B^+$ such that both ρ_1 and ρ_2 are productive, weakly-balanced and $x-$commuting. The same holds for the other three cases where exactly two of the four components of e_1 and e_2 are empty.

If $e_1 \in B^+ \times \{\varepsilon\}$ and $e_2 \in B^+ \times B^+$, then there exist $g_1, g_2 \in \mathcal{C}(B)$ and i_0 such that there exist infinitely many integers $i \geqslant i_0$, such that $g_1[\overleftarrow{e_1}^{i-i_0}] = g_2 e_2^{i-i_0} d_2 c_2[\varepsilon]$. Therefore, by Lemma 5.2, we obtain that for all $i \in \mathbb{N}$, $g_1[\overleftarrow{e_1}^{i}] = g_2 e_2^i d_2 c_2[\varepsilon]$. Then

by Lemma 5.15, there exists $x \in B^+$ such that both ρ_1 and ρ_2 are productive, weakly-balanced and $x-$commuting. The same holds for the other three cases where exactly one of the four components of e_1 and e_2 is empty.

If $e_1, e_2 \in B^+ \times B^+$, then there exist $g_1, g_2 \in \mathcal{C}(B)$ and i_0 such that there exist infinitely many integers $i \geqslant i_0$, such that $g_1 e_1^{i-i_0} d_1 c_1[\varepsilon] = g_2 e_2^{i-i_0} d_2 c_2[\varepsilon]$. Therefore, by Lemma 5.2, we obtain that for all $i \in \mathbb{N}$, $g_1 e_1^i d_1 c_1[\varepsilon] = g_2 e_2^i d_2 c_2[\varepsilon]$. Then by Lemma 5.16, there are two cases. Firstly, if $\|e_1\| \neq \|e_2\|$ then there exists $x \in B^+$ such that both ρ_1 and ρ_2 are productive, weakly-balanced and $x-$commuting. Secondly, if $\|e_1\| = \|e_2\|$ then there exist $f \in \mathcal{C}(B)$ and $w \in B^*$ such that both ρ_1 and ρ_2 are productive, strongly-balanced, and $(f,w)-$aligned. However, by Lemma 5.17, if it still happens that either one of ρ_1 and ρ_2 is $x-$commuting, then both ρ_1 and ρ_2 are $x-$commuting. If not, then they both are non-commuting. □

5.3.1.2 Lifting to k Synchronised Lassos

Lemma 5.18. *Let $x \in B^+$ a primitive word. Let ρ_1 and ρ_2 be two synchronised productive lassos. If ρ_1 is $x-$commuting, then ρ_2 is $x-$commuting.*

Proof. Let $\rho_1 : \xrightarrow{c_1} p_1 \xrightarrow{u_1|d_1} q_1 \xrightarrow{u_2|e_1} q_1$ and $\rho_2 : \xrightarrow{c_2} p_2 \xrightarrow{u_1|d_2} q_2 \xrightarrow{u_2|e_2} q_2$. By Lemma 5.12, we observe two cases. First, consider that $\|e_1\| = \|e_2\|$ and that there exists $f \in \mathcal{C}(B)$ and $w \in B^*$ such that ρ_1 and ρ_2 are $(f,w)-$aligned. By Lemma 5.17, if ρ_1 is $x-$commuting, then ρ_2 is $x-$commuting. Second, we only have that $|e_1| = |e_2|$, and there exists $x' \in B^+$ primitive such that ρ_1 and ρ_2 are $x'-$commuting. If ρ_1 is also $x-$commuting then, thanks to Lemma 1.1, we can prove that $x' \sim x$ and then that ρ_2 is also $x-$commuting. □

The following lemma states a consequence of the definition of aligned lassos.

Lemma 5.19. *Let ρ be a productive lasso $\xrightarrow{c} p \xrightarrow{u|d} q \xrightarrow{v|e} q$ and $f \in \mathcal{C}(B)$ and $w \in B^*$. ρ is $(f,w)-$aligned if and only if there exist t_1, t_2, t_3, t_4 and $\alpha, \beta \geqslant 0$ such that $\mathsf{root}(\overleftarrow{e}) = t_1 t_2$, $\mathsf{root}(\overleftarrow{f}) = t_2 t_1$, $\mathsf{root}(\overrightarrow{e}) = t_3 t_4$, $\mathsf{root}(\overrightarrow{f}) = t_4 t_3$, and $dc[\varepsilon] = (t_1 t_2)^\alpha t_1 w t_3 (t_4 t_3)^\beta$.*

As a corollary, any productive lasso $\rho : \xrightarrow{c} p \xrightarrow{u|d} q \xrightarrow{v|e} q$ is $(e, dc[\varepsilon])-$aligned. Also, note that a lasso can be commuting and aligned at the same time.

Lemma 5.20. *Let $\rho_1, \ldots, \rho_k$ be k synchronised productive lassos that are pairwise aligned, strongly balanced and not commuting. Then there exist $f \in \mathcal{C}(B)$ and $w \in B^*$ such that they are all $(f,w)-$aligned.*

Proof. Let $\rho_i : \xrightarrow{c_i} p_i \xrightarrow{u_1|d_i} q_i \xrightarrow{u_2|e_i} q_i$ for $i \in \{1, \ldots, k\}$. As $\rho_1, \ldots, \rho_k$ are pairwise aligned, there exist $f_2, \ldots, f_k \in \mathcal{C}(B)$ and $w_2, \ldots, w_k \in B^*$ such that for all $i \in \{2, \ldots, k\}$, ρ_1 and ρ_i are $(f_i, w_i)-$aligned. Then for all $i \in \{2, \ldots, k\}$, there exist

$f_i, g_i, h_i \in \mathcal{C}(B)$ and $w_i \in B^*$ such that for all $j \in \mathbb{N}$, $e_1^j d_1 c_1[\varepsilon] = g_i f_i^j[w_i]$ and $e_i^j d_i c_i[\varepsilon] = h_i f_i^j[w_i]$.

Let $\ell, r \in \{2, \dots, k\}$ such that $|\overleftarrow{g_\ell}| = max\{|\overleftarrow{g_i}| \mid i \in \{2, \dots, k\}\}$ and $|\overrightarrow{g_r}| = max\{|\overrightarrow{g_i}| \mid i \in \{2, \dots, k\}\}$. Let $g = (\overleftarrow{g_\ell}, \overrightarrow{g_r})$, $f = (\overleftarrow{f_\ell}, \overrightarrow{f_r})$, and $w = g^{-1} d_1 c_1[\varepsilon]$. By definition, for all $i \in \{2, \dots, k\}$, $|\overleftarrow{g_i}| \leqslant |\overleftarrow{g_\ell}|$ and $|\overleftarrow{g_i}| \leqslant |\overleftarrow{g_r}|$. Thus, for all $i \in \{2, \dots, k\}$, $g_i^{-1} g \in \mathcal{C}(B)$. We have that $|g| > |d_1 c_1|$, otherwise it would contradict that the lassos are all non-commuting. Thus, $w \in B^*$.

By Lemma 5.19, we have that $\mathsf{root}(\overleftarrow{e}) \sim \mathsf{root}(\overleftarrow{f_\ell})$ and $\mathsf{root}(\overrightarrow{e}) \sim \mathsf{root}(\overrightarrow{f_r})$. Therefore, we can show that for all $j \in \mathbb{N}$, $e_1^j d_1 c_1[\varepsilon] = g f^j[w]$. Then, for all $i \in \{2, \dots, k\}$ and $j \in \mathbb{N}$, $e_i^j d_i c_i[\varepsilon] = h_i f_i^j[w_i] = h_i g_i^{-1} e_1^j d_1 c_1[\varepsilon] = h_i g_i^{-1} g f^j[w]$. □

We are finally ready to prove Lemma 5.11.

Proof of Lemma 5.11. The length of the contexts labelling the loops must be equal, as the outputs must grow at the same pace when the loops are pumped. By Lemma 5.18, if one of the lassos is $x-\mathrm{commuting}$ then they are all $x-\mathrm{commuting}$. Otherwise, none of them are commuting. Then, by Lemma 5.12, they are also all pairwise aligned and strongly balanced. Therefore by Lemma 5.20, there exists $f \in \mathcal{C}(B)$ and $w \in B^*$ such that they are all $(f, w)-\mathrm{aligned}$. □

5.3.2 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 $x-\mathrm{commuting}$ lasso cannot be followed by a $y-\mathrm{commuting}$ 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 5.5 (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 by $\mathrm{strongly}-x-\mathrm{commuting}$, if there exists a context $f \in \mathcal{C}(B)$ such that for all $i > 0$, $j > 0$, 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 by $\mathrm{strongly}-(g, f, x)-\mathrm{aligned}$, if there exists a context $h \in \mathcal{C}(B)$ such that for all $i, j \in \mathbb{N}$, $e^j dc[x^i] = h g^j f[x^i]$.

The following lemma states the properties of a lasso consecutive to a commuting lasso.

Lemma 5.21. *Let $H_1 H_2$ a productive $x-\mathrm{commuting}$ lasso in $\mathcal{T}^{\leqslant |Q|}$, for some $x \in B^+$. Let $\Delta = \mathsf{split}_c(x, H_1, H_2)$ and $H_3 H_4 = (\rho_j)_{j \in \{1, \dots, k\}}$ 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* $j \in\in \{1, \dots, k\}$. *Then:*

— *either* ρ_j *is* $\mathrm{strongly}{-}x{-}\mathrm{commuting}$ *for all* $j \in \{1, \dots, k\}$. *In this case, we say that the lasso* H_3H_4 *is* $\mathrm{strongly}{-}x{-}\mathrm{commuting}$.

— *or there exist* $g, h \in \mathcal{C}(B)$ *such that* ρ_j *is* $\mathrm{strongly}{-}(h, g, x){-}\mathrm{aligned}$ *for all* $j \in \{1, \dots, k\}$. *In this case, we say that* H_3H_4 *is* $\mathrm{strongly}{-}(h, g, x){-}\mathrm{aligned}$ *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 FST derived from the S2C both satisfy the twinning property:

Lemma 5.22. *Let* H_1H_2 *be a productive non-commuting lasso that is*

— *either* $(f, w){-}\mathrm{aligned}$ *in* $\mathcal{T}^{\leqslant |Q|}$, *for some* $f \in \mathcal{C}(B)$ *and* $w \in B^*$, *and we let* $\Delta' = \mathsf{split}_{nc}(f, w, H_1, H_2)$,

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

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

In the remainder of this subsection, we will prove Lemmas 5.21 and 5.22.

5.3.2.1 Lassos Consecutive to a Commuting Lasso

In order to prove Lemma 5.21, we proceed as for Lemma 5.11 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 1.1.

We state the following lemma for two runs.

Lemma 5.23. *Let* $x \in B^+$ *a primitive word and let* $\Delta = \mathsf{split}_c(x, H_1, H_2)$ *for some* H_1H_2 *an* $x{-}\mathrm{commuting}$ *lasso in* $\mathcal{T}^k$. *For any two synchronised lassos* ρ_1 *and* ρ_2 *in* $\mathcal{T}_\Delta$, *we have that*

— *either* ρ_1 *and* ρ_2 *are non-productive,*

— *or* ρ_1 *and* ρ_2 *are productive, weakly-balanced, and* $\mathrm{strongly}{-}x{-}\mathrm{commuting}$,

— *or* ρ_1 *and* ρ_2 *are productive, strongly-balanced, non-commuting, and there exists* $g, f \in \mathcal{C}(B)$ *such that* ρ_1 *and* ρ_2 *are* $\mathrm{strongly}{-}(g, f, x){-}\mathrm{aligned}$.

In order to prove Lemma 5.23, we first need some additional combinatorial results.

Lemma 5.24. *Let* $c_1, c_2, d_1, d_2 \in \mathcal{C}(B)$ *and* $x \in B^+$ *a primitive word. If for all* $i, j \in \mathbb{N}$, $\mathit{dist}_f(d_1^j c_1[x^i], d_2^j c_2[x^i]) \leqslant L$, *then there exist* $e_1, e_2 \in \mathcal{C}(B)$ *and a set* $I \subseteq \mathbb{N} \times \mathbb{N}$ *such that for all* $(i, j) \in I$, $e_1^{-1} d_1^j c_1[x^i] = e_2^{-1} d_2^j c_2[x^i]$, *and for all* $(i_0, j_0) \in \mathbb{N} \times \mathbb{N}$, *there exists* $(i, j) \in I$ *such that* $i > i_0$ *and* $j > j_0$.

Proof. Suppose that for all $i, j \in \mathbb{N}$, $\mathsf{dist}_f(d_1^j c_1[x^i], d_2^j c_2[x^i]) \leqslant L$. Then for all $i, j \in \mathbb{N}$, there exist $f_1, f_2 \in \mathcal{C}(B)$, such that $f_1^{-1} d_1^j c_1[x^i] = f_2^{-1} d_2^j c_2[x^i]$ and $|f_1| + |f_2| \leqslant L$. Let $C_L = \{(f_1, f_2) \mid |f_1| + |f_2| \leqslant L\}$. C_L is finite. Therefore, we can prove that there exist some $(e_1, e_2) \in C_L$ and a set $I \subseteq \mathbb{N} \times \mathbb{N}$ such that for all $(i, j) \in I$, $e_1^{-1} d_1^j c_1[x^i] = e_2^{-1} d_2^j c_2[x^i]$, and for all $(i_0, j_0) \in \mathbb{N} \times \mathbb{N}$, there exists $(i, j) \in I$ such that $i > i_0$ and $j > j_0$. □

Lemma 5.25. *Let $s_1, u_1, w_1, u_2, w_2, y_2 \in B^*$ and $t_1, x_2, v \in B^+$ such that v is primitive, $|t_1| = |x_2|$ and for all $i, j \in \mathbb{N}$, $s_1 t_1^j u_1 v^i w_1 = u_2 v^i w_2 x_2^j y_2$. Then there exist $f_1, f_2 \in \mathcal{C}(B)$ such that for all $i, j \geqslant 1$ there exists $k \in \mathbb{N}$ such that $t_1^j u_1 v^i = f_1[v^k]$ and $v^i w_2 x_2^j = f_2[v^k]$.*

Proof. We can find sufficiently large i_0 and j_0 such that $t_1^{j_0}$ and $x_2^{j_0}$ both overlap with v^{i_0} with a common factor of length greater than $|t_1| + |v| - \mathsf{gcd}(|t_1|, |v|) = |x_2| + |v| - \mathsf{gcd}(|x_2|, |v|)$. Thus by Lemma 1.1, $\mathsf{root}(t_1) \sim \mathsf{root}(x_2) \sim \mathsf{root}(v)$. As $|t_1| = |x_2|$, $|t_1 u_1 v| \geqslant |x_2|$. Yet $t_1 u_1 v$ is a factor of x_2^* and, by Lemma 5.1, $\mathsf{period}(t_1 u_1 v) \sim \mathsf{root}(x_2)$. Similarly, $\mathsf{period}(v w_2 x_2) \sim \mathsf{root}(t_1)$. Thus $\mathsf{root}(t_1) \sim \mathsf{root}(x_2) \sim \mathsf{period}(t_1 u_1 v) \sim \mathsf{period}(v w_2 x_2) \sim \mathsf{root}(v)$.

Let $z_1, z_1', z_2, z_2' \in B^*$ and $\alpha \geqslant 1$ such that $z_1 z_1' = z_2 z_2'$, and $v = z_1 z_1'$, $t_1 = (z_1' z_1)^\alpha$ and $x_2 = (z_2' z_2)^\alpha$. Also we have $t_1 u_1 v = z_1'(z_1 z_1')^{\theta_1}$, $v w_2 x_2 = (z_1 z_1')^{\theta_2} z_2$, for some $\theta_1, \theta_2 \geqslant 0$.

We choose $f_1 = (z_1', (z_1 z_1')^{\theta_1})$, and $f_2 = ((z_1 z_1')^{\theta_2}, z_2)$. Then, for all $i, j \geqslant 1$, we have $t_1^j u_1 v^i = z_1'(z_1 z_1')^{\alpha(j-1)+\theta_1+(i-1)}$ and $v^i w_2 x_2^j = (z_1 z_1')^{(i-1)+\theta_2+\alpha(j-1)} z_2$. Let $k = (i-1) + \alpha(j-1) \geqslant 0$. And we obtain $t_1^j u_1 v^i = f_1[v^k]$ and $v^i w_2 x_2^j = f_2[v^k]$. □

Lemma 5.26. *Let $s_1, u_1, w_1, s_2, u_2, w_2 \in B^*$ and $t_1, t_2, v \in B^+$ such that $|t_1| = |t_2|$ and for all $i, j \in \mathbb{N}$, $s_1 t_1^j u_1 v^i w_1 = s_2 t_2^j u_2 v^i w_2$. Then there exist some contexts $f, g, h_1, h_2 \in \mathcal{C}(B)$ such that for all $i, j \in \mathbb{N}$, $t_1^j u_1 v^i = h_1 g^j f[v^i]$ and $t_2^j u_2 v^i = h_2 g^j f[v^i]$.*

Proof. There exists j_0 sufficiently large such that $t_1^{j_0}$ overlap with $t_2^{j_0}$ with a common factor of length greater than $|t_1| + |t_2| - \mathsf{gcd}(|t_1|, |t_2|) = |t_1| = |t_2|$. Thus by Lemma 1.1, $\mathsf{root}(t_1) \sim \mathsf{root}(t_2)$. If $|s_1| \leqslant |s_2|$, let t such that $s_1 t = s_2$ and $t_1 t = t t_2$. If $|s_1| \geqslant |s_2|$, let t such that $s_1 = s_2 t$ and $t t_1 = t_2 t$. If $|w_1| \leqslant |w_2|$, let i_0 such that $v^{i_0} w_1 = w_2$ and v^{i_0} is a suffix of u_1. If $|w_1| \geqslant |w_2|$, let i_0 such that $w_1 = v^{i_0} w_2$ and v^{i_0} is a suffix of u_2. We obtain that

— $s_1 t = s_2$, $t_1 t = t t_2$, $u_1 = t u_2 v^{i_0}$, $v^{i_0} w_1 = w_2$, or
— $s_1 = s_2 t$, $t t_1 = t_2 t$, $t u_1 = u_2 v^{i_0}$, $v^{i_0} w_1 = w_2$, or
— $s_1 t = s_2$, $t_1 t = t t_2$, $u_1 v^{i_0} = t u_2$, $w_1 = v^{i_0} w_2$, or
— $s_1 = s_2 t$, $t t_1 = t_2 t$, $t u_1 v^{i_0} = u_2$, $w_1 = v^{i_0} w_2$.

We handle the first case. The others are similar. We choose $h_1 = (t, v^{i_0})$, $h_2 = (\varepsilon, \varepsilon)$, $g = (t_2, \varepsilon)$, and $f = (u_2, \varepsilon)$. Then for all $i \in \mathbb{N}$, $t_1^j u_1 v^i = t_1^j t u_2 v^{i_0} v^i = t t_2^j u_2 v^i v^{i_0} = h_1 g^j f[v^i]$ and $t_2^j u_2 v^i = h_2 g^j f[v^i]$. □

Lemma 5.27. *Let $s_1, u_1, w_1, s_2, u_2, w_2, y_2 \in B^*$ and $t_1, t_2, x_2, v \in B^+$ such that $|t_1| = |t_2| + |x_2|$ and for all $i, j \in \mathbb{N}$, $s_1 t_1^j u_1 v^i w_1 = s_2 t_2^j u_2 v^i w_2 x_2^j y_2$. Then there exist $f_1, f_2 \in \mathcal{C}(B)$ such that for all $i, j \geqslant 1$ there exists $k \in \mathbb{N}$ such that $t_1^j u_1 v^i = f_1[v^k]$ and $t_2^j u_2 v^i w_2 x_2^j = f_2[v^k]$.*

Proof. We can find sufficiently large i_0 and j_0 such that $t_1^{j_0}$ overlap with $t_2^{j_0}$, v^{i_0} and $x_2^{j_0}$ with common factors of respective length greater than $|t_1| + |t_2| - \mathsf{gcd}(|t_1|, |t_2|)$, $|t_1| + |v| - \mathsf{gcd}(|t_1|, |v|)$, and $|t_1| + |x_2| - \mathsf{gcd}(|t_1|, |x_2|)$. Thus by Lemma 1.1, $\mathsf{root}(t_1) \sim \mathsf{root}(t_2) \sim \mathsf{root}(x_2) \sim \mathsf{root}(v)$. As $|t_1| = |x_2|$, $|t_1 u_1 v| \geqslant |x_2|$. Similarly to Lemma 5.25, we can show that $\mathsf{period}(t_1 u_1 v) \sim \mathsf{period}(t_2 u_2 v) \sim \mathsf{period}(v w_2 x_2) \sim \mathsf{root}(v)$, and reconstruct the words to obtain the result. □

Lemma 5.28. *Let $s_1, u_1, w_1, y_1, s_2, u_2, w_2, y_2 \in B^*$ and $t_1, x_1, t_2, x_2, v \in B^+$ such that $|t_1| + |t_2| = |x_1| + |x_2|$, $|t_1| \neq |x_1|$ and $|t_2| \neq |x_2|$, and for all $i, j \in \mathbb{N}$, $s_1 t_1^j u_1 v^i w_1 x_1^j y_1 = s_2 t_2^j u_2 v^i w_2 x_2^j y_2$. Then there exist $f_1, f_2 \in \mathcal{C}(B)$ such that for all $i, j \geqslant 1$ there exists $k \in \mathbb{N}$ such that $t_1^j u_1 v^i w_1 x_1^j = f_1[v^k]$ and $t_2^j u_2 v^i w_2 x_2^j = f_2[v^k]$.*

Proof. Without loss of generality, consider that $|t_1| > |t_2|$. We can find sufficiently large i_0 and j_0 such that $t_1^{j_0}$ overlap with $t_2^{j_0}$, v^{i_0} and $x_2^{j_0}$ with common factors of respective length greater than $|t_1| + |t_2| - \mathsf{gcd}(|t_1|, |t_2|)$, $|t_1| + |v| - \mathsf{gcd}(|t_1|, |v|)$, and $|t_1| + |x_2| - \mathsf{gcd}(|t_1|, |x_2|)$. Thus by Lemma 1.1, $\mathsf{root}(t_1) \sim \mathsf{root}(t_2) \sim \mathsf{root}(x_2) \sim \mathsf{root}(v)$. Similarly, we can show that $\mathsf{root}(x_1) \sim \mathsf{root}(x_2)$. Finally, similarly to Lemma 5.25, we can show that $\mathsf{period}(t_1 u_1 v) \sim \mathsf{period}(t_2 u_2 v) \sim \mathsf{period}(v w_1 x_1) \sim \mathsf{period}(v w_2 x_2) \sim \mathsf{root}(v)$, and reconstruct the words to obtain the result. □

Lemma 5.29. *Let $s_1, u_1, w_1, y_1, s_2, u_2, w_2, y_2 \in B^*$ and $t_1, x_1, t_2, x_2, v \in B^+$ such that $|t_1| = |x_1|$, $|t_2| = |x_2|$, and for all $i, j \in \mathbb{N}$, $s_1 t_1^j u_1 v^i w_1 x_1^j y_1 = s_2 t_2^j u_2 v^i w_2 x_2^j y_2$. Then there exist some contexts $f, g, h_1, h_2 \in \mathcal{C}(B)$ such that for all $i, j \in \mathbb{N}$, we have $t_1^j u_1 v^i w_1 x_1^j = h_1 g^j f[v^i]$ and $t_2^j u_2 v^i w_2 x_2^j = h_2 g^j f[v^i]$.*

Proof. There exists j_0 sufficiently large such that $t_1^{j_0}$ overlap with $t_2^{j_0}$ with a common factor of length greater than $|t_1| + |t_2| - \mathsf{gcd}(|t_1|, |t_2|) = |t_1| = |t_2|$. Thus by Lemma 1.1, $\mathsf{root}(t_1) \sim \mathsf{root}(t_2)$. The same applies to $x_1^{j_0}$ and $x_2^{j_0}$, and $\mathsf{root}(x_1) \sim \mathsf{root}(x_2)$. By a reasoning similar to Lemma 5.26, based on the length of s_1, s_2, y_1 and y_2, we can show that there exists $t, x \in B^*$ such that we obtain

- $s_1 t = s_2$, $t_1 t = t t_2$, $u_1 v^i w_1 = t u_2 v^i w_2 x$, $x x_1 = x_2 x$, $x y_1 = y_2$, or
- $s_1 t = s_2$, $t_1 t = t t_2$, $u_1 v^i w_1 x = t u_2 v^i w_2$, $x_1 x = x x_2$, $y_1 = x y_2$, or
- $s_1 = s_2 t$, $t t_1 = t_2 t$, $t u_1 v^i w_1 = u_2 v^i w_2 x$, $x x_1 = x_2 x$, $x y_1 = y_2$, or
- $s_1 = s_2 t$, $t t_1 = t_2 t$, $t u_1 v^i w_1 x = u_2 v^i w_2$, $x_1 x = x x_2$, $y_1 = x y_2$.

Again similarly to Lemma 5.26, we can reconstruct the words $t_1^j u_1 v^i w_1 x_1^j$ and $t_2^j u_2 v^i w_2 x_2^j$ to obtain the result. □

Proof of Lemma 5.23. Let $x \in B^+$ a primitive word, $H_1 H_2$ be an $x-$commuting in $\mathcal{T}^k$, and let $\Delta = \mathsf{split}_c(x, H_1, H_2)$. Let $\rho_1 : \xrightarrow{c_1} p_1 \xrightarrow{u_1|d_1} q_1 \xrightarrow{u_2|e_1} q_1$ and $\rho_2 : \xrightarrow{c_2} p_2 \xrightarrow{u_1|d_2} q_2 \xrightarrow{u_2|e_2} q_2$ be two lassos in $\mathcal{T}_\Delta$.

As H_1H_2 is $x-$commuting, we can find two synchronised $x-$commuting lassos in $\mathcal{T}$ around p_1 and p_2. Let $\rho_1' : \xrightarrow{c_{0,1}} i_1 \xrightarrow{t_1|c_{1,1}} p_1 \xrightarrow{t_2|c_{2,1}} p_1$ and $\rho_2' : \xrightarrow{c_{0,2}} i_2 \xrightarrow{t_1|c_{1,2}} p_2 \xrightarrow{t_2|c_{2,2}} p_2$ be those two lassos. Furthermore, by definition of split_c, we have that $\Delta(p_1) = c_1$ and $\Delta(p_2) = c_2$, and for all $i \in \mathbb{N}$, there exists $k \in \mathbb{N}$ such that $c^i_{2,1}c_{1,1}c_{0,1}[\varepsilon] = c_1[x^k]$ and $c^i_{2,2}c_{1,2}c_{0,2}[\varepsilon] = c_2[x^k]$. By Definition 5.3, for all $i, j \in \mathbb{N}$, $\mathsf{dist}_f(e_1^j d_1 c_1[x^i], e_2^j d_2 c_2[x^i]) \leqslant L$.

By Lemma 5.24, there exist $f_1, f_2 \in \mathcal{C}(B)$ and a set $I \subseteq \mathbb{N} \times \mathbb{N}$ such that for all $(i, j) \in I$, $f_1^{-1} e_1^j d_1 c_1[x^i] = f_2^{-1} e_2^j d_2 c_2[x^i]$, and for all $(i_0, j_0) \in \mathbb{N} \times \mathbb{N}$, there exists $(i, j) \in I$ such that $i > i_0$ and $j > j_0$. Then we have that $|e_1| = |e_2|$. We observe 10 cases.

If $|e_1| = 0$ or $|e_2| = 0$ then $|e_1| = |e_2| = 0$ and ρ_1, ρ_2 are non-productive.

If $e_1 \in \{\varepsilon\} \times B^+$ and $e_2 \in B^+ \times \{\varepsilon\}$, then there exists $g_1, g_2 \in \mathcal{C}(B)$ and $(i_0, j_0) \in \mathbb{N} \times \mathbb{N}$ such that for all $(i, j) \in I$ such that $i \geqslant i_0$ and $j \geqslant j_0$, $g_1[x^{i-i_0}\overrightarrow{e_1}^{j-j_0}\overrightarrow{d_1c_1}] = g_2[\overleftarrow{e_2}^{j-j_0}\overleftarrow{d_2c_2}x^{i-i_0}]$. Therefore, by applying Lemma 5.2 two times, we obtain that for all $i, j \in \mathbb{N}$, $g_1[x^i\overrightarrow{e_1}^j\overrightarrow{d_1c_1}] = g_2[\overleftarrow{e_2}^j\overleftarrow{d_2c_2}x^i]$. Then by Lemma 5.25, both ρ_1 and ρ_2 are productive and $\mathrm{strongly}-x-\mathrm{commuting}$. The same holds for the symmetrical case where $e_1 \in B^+ \times \{\varepsilon\}$ and $e_2\{\varepsilon\} \times B^+$.

If $e_1, e_2 \in B^+ \times \{\varepsilon\}$, then there exists $g_1, g_2 \in \mathcal{C}(B)$ and $(i_0, j_0) \in \mathbb{N} \times \mathbb{N}$ such that for all $(i, j) \in I$ such that $i \geqslant i_0$ and $j \geqslant j_0$, $g_1[\overleftarrow{e_1}^{j-j_0}\overleftarrow{d_1c_1}x^{i-i_0}] = g_2[\overleftarrow{e_2}^{j-j_0}\overleftarrow{d_2c_2}x^{i-i_0}]$. Therefore, by applying Lemma 5.2 two times, we obtain that for all $i, j \in \mathbb{N}$, $g_1[\overleftarrow{e_1}^j\overleftarrow{d_1c_1}x^i] = g_2[\overleftarrow{e_2}^j\overleftarrow{d_2c_2}x^i]$. Then by Lemma 5.26, there exist $f, g \in \mathcal{C}(B)$ such that both ρ_1 and ρ_2 are productive and $\mathrm{strongly}-(g, f, x)-\mathrm{aligned}$. However, if it still happens that either one of ρ_1 and ρ_2 is $\mathrm{strongly}-x-\mathrm{commuting}$ (implying that $\mathsf{root}(\overleftarrow{e_1}) \sim \mathsf{root}(\overrightarrow{e_1}) \sim x$ or $\mathsf{root}(\overleftarrow{e_2}) \sim \mathsf{root}(\overrightarrow{e_2}) \sim x$), then both ρ_1 and ρ_2 are $\mathrm{strongly}-x-\mathrm{commuting}$. If not, then they both are non-commuting. The same holds for the symmetrical case where $e_1, e_2 \in \{\varepsilon\} \times B^+$.

If $e_1 \in B^+ \times \{\varepsilon\}$ and $e_2 \in B^+ \times B^+$, then there exists $g_1, g_2 \in \mathcal{C}(B)$ and $(i_0, j_0) \in \mathbb{N} \times \mathbb{N}$ such that for all $(i, j) \in I$ such that $i \geqslant i_0$ and $j \geqslant j_0$, $g_1[\overleftarrow{e_1}^{j-j_0}\overleftarrow{d_1c_1}x^{i-i_0}] = g_2e_2^{j-j_0}d_2c_2[x^{i-i_0}]$. Therefore, by applying Lemma 5.2 two times, we obtain that for all $i, j \in \mathbb{N}$, $g_1[\overleftarrow{e_1}^j\overleftarrow{d_1c_1}x^i] = g_2e_2^jd_2c_2[x^i]$. Then by Lemma 5.27, both ρ_1 and ρ_2 are productive and $\mathrm{strongly}-x-\mathrm{commuting}$. The same holds for the other three cases where exactly one of the four components of e_1 and e_2 is empty.

If $e_1, e_2 \in B^+ \times B^+$ and $\|e_1\| \neq \|e_2\|$, then there exists $g_1, g_2 \in \mathcal{C}(B)$ and $(i_0, j_0) \in \mathbb{N} \times \mathbb{N}$ such that for all $(i, j) \in I$ such that $i \geqslant i_0$ and $j \geqslant j_0$, $g_1e_1^{j-j_0}d_1c_1[x^{i-i_0}] = g_2e_2^{j-j_0}d_2c_2[x^{i-i_0}]$. Therefore, by applying Lemma 5.2 two times, we obtain that for all $i, j \in \mathbb{N}$, $g_1e_1^jd_1c_1[x^i] = g_2e_2^jd_2c_2[x^i]$. Then by Lemma 5.28, both ρ_1 and ρ_2 are productive and $\mathrm{strongly}-x-\mathrm{commuting}$.

If $e_1, e_2 \in B^+ \times B^+$ and $\|e_1\| = \|e_2\|$, then there exists $g_1, g_2 \in \mathcal{C}(B)$ and $(i_0, j_0) \in \mathbb{N} \times \mathbb{N}$ such that for all $(i, j) \in I$ such that $i \geqslant i_0$ and $j \geqslant j_0$, $g_1e_1^{j-j_0}d_1c_1[x^{i-i_0}] =$

$g_2 e_2^{j-j_0} d_2 c_2[x^{i-i_0}]$. Therefore, by applying Lemma 5.2 two times, we obtain that for all $i, j \in \mathbb{N}$, $g_1 e_1^j d_1 c_1[x^i] = g_2 e_2^j d_2 c_2[x^i]$. Then by Lemma 5.29, there exist $f, g \in \mathcal{C}(B)$ such that both ρ_1 and ρ_2 are productive and $\mathrm{strongly}{-}(g, f, x){-}\mathrm{aligned}$. However, if it still happens that either one of ρ_1 and ρ_2 is $\mathrm{strongly}{-}x{-}\mathrm{commuting}$ (implying that $\mathsf{root}(\overleftarrow{e_1}) \sim \mathsf{root}(\overrightarrow{e_1}) \sim x$ or $\mathsf{root}(\overleftarrow{e_2}) \sim \mathsf{root}(\overrightarrow{e_2}) \sim x$), then both ρ_1 and ρ_2 are $\mathrm{strongly}{-}x{-}\mathrm{commuting}$. If not, then they both are non-commuting. □

Proof of Lemma 5.21. By Lemma 5.23, if not productive, two lassos following a commuting lasso are either strongly commuting or strongly aligned. This result can be lifted to k runs in a similar way to Lemma 5.11. □

5.3.2.2 Lassos Consecutive to a Non-Commuting Lasso

We finally study the properties of lassos that are consecutive to a lasso that is not commuting, in order to prove Lemma 5.22. The following lemma shows that only a non-commuting lasso can follow a non-commuting lasso.

Lemma 5.30. *Let $f \in \mathcal{C}(B)$, $w \in B^*$, such that there exists no $x \in B^+$ and $g \in \mathcal{C}(B)$ such that, for all $i \in \mathbb{N}$, there exists $k \in \mathbb{N}$ such that $f^i[w] = g[x^k]$. Let $c, d \in \mathcal{C}(B)$, and $i \geqslant 1$. Then there exists no $x \in B^+$ and $g \in \mathcal{C}(B)$ such that, for all $j \in \mathbb{N}$, there exists $k \in \mathbb{N}$ such that $d^j c f^i[w] = g[x^k]$.*

Proof. This can easily be proved by contradiction. □

The lassos after a non-commuting lasso are then always fully-aligned. Given a string-to-context transducer that satisfies the contextual twinning property, we can thus view its restriction after an aligned lasso as the pair of two classical finite-state transducers that both satisfy the classical twinning property.

Lemma 5.31. *Let $f \in \mathcal{C}(B)$ and $w \in B^*$ and let $\Delta = \mathsf{split}_{nc}(f, w, H_1, H_2)$, for some $H_1 H_2$ a productive, non-commuting and $(f, w) - aligned$ lasso in $\mathcal{T}^{\leqslant |Q|}$. Then $\overrightarrow{\mathcal{T}_\Delta}$ and $\overleftarrow{\mathcal{T}_\Delta}$ both satisfy the twinning property.*

Proof. We show the result for $\overrightarrow{\mathcal{T}_\Delta}$. The proof for $\overleftarrow{\mathcal{T}_\Delta}$ is symmetrical.

Let $\xrightarrow{x} p \xrightarrow{u|y} q \xrightarrow{v|z} q$ and $\xrightarrow{x'} p' \xrightarrow{u|y'} q' \xrightarrow{v|z'} q'$ two lassos in $\overrightarrow{\mathcal{T}_\Delta}$. By the definition of $\overrightarrow{\mathcal{T}_\Delta}$, there exist $(p, c), (p', c') \in \Delta$ such that $\overrightarrow{c} = x$ and $\overrightarrow{c'} = x'$, and $p \xrightarrow{u|d} q \xrightarrow{v|e} q$ and $p' \xrightarrow{u|d'} q' \xrightarrow{v|e'} q'$ in $\mathcal{T}_\Delta$ such that $\overrightarrow{d} = y$, $\overrightarrow{d'} = y'$, $\overrightarrow{e} = z$, $\overrightarrow{e'} = z'$.

From $\Delta = \mathsf{split}_{nc}(f, w, H_1, H_2)$ and $H_1 H_2$ being non-commuting, we know that there exist $i \geqslant 1$ and two lassos $\xrightarrow{c_1} o \xrightarrow{s|c_2} p \xrightarrow{t|c_3} p$ and $\xrightarrow{c'_1} o' \xrightarrow{s|c'_2} p' \xrightarrow{t|c'_3} p'$ in $\mathcal{T}$, such that $c_3 c_2 c_1[\varepsilon] = c f^i[w]$ and $c'_3 c'_2 c'_1[\varepsilon] = c' f^i[w]$. Thus we can build two lassos in $\mathcal{T}$: $\rho_1 : \xrightarrow{c_1} o \xrightarrow{s|dc_3c_2} q \xrightarrow{v|e} q$ and $\rho_2 : \xrightarrow{c'_1} o' \xrightarrow{s|d'c'_3c'_2} q' \xrightarrow{v|e'} q'$.

By Definition 5.3, for all $j \in \mathbb{N}$, $\mathsf{dist}_f(e^j dc f^i[w], e'^j d' c' f^i[w]) \leqslant L$. As $H_1 H_2$ is non-commuting, by Lemma 5.30, ρ_1 and ρ_2 must also be non-commuting, and

thus strongly-balanced. As H_1H_2 and ρ_1 and ρ_2 are strongly-balanced and non-commuting the $f^i[w]$ part can only overlap with itself in the words $e^j dc f^i[w]$ and $e'^j d'c' f^i[w]$. Therefore, we can derive that for all $j \in \mathbb{N}$, $\mathsf{dist}_p(xyz^j, x'y'z'^j) \leqslant L$. □

Lemma 5.32. *Let $x \in B^+$ a primitive word and let $\Delta_0 = \mathsf{split}_c(x, H_1, H_2)$, for some H_1H_2 a productive and $x-$commuting lasso in $\mathcal{T}^{\leqslant |Q|}$. Let $g, f \in \mathcal{C}(B)$ and let $\Delta = \mathsf{extract}_{nc}(g, f, x, \Delta_0, H_3, H_4)$, for some H_3H_4 a productive, non-commuting and strongly$-(g,f,x)-$aligned lasso in $\mathcal{T}^{\leqslant |Q|}_{\Delta_0}$. Then $\overrightarrow{\mathcal{T}_\Delta}$ and $\overleftarrow{\mathcal{T}_\Delta}$ both satisfy the twinning property.*

Proof. We show the result for $\overrightarrow{\mathcal{T}_\Delta}$. The proof for $\overleftarrow{\mathcal{T}_\Delta}$ is symmetrical.

Let $\xrightarrow{v_0} q_2 \xrightarrow{u_1|v_1} q_3 \xrightarrow{u_2|v_2} q_3$ and $\xrightarrow{v'_0} q'_2 \xrightarrow{u_1|v'_1} q'_3 \xrightarrow{u_2|v'_2} q'_3$ two lassos in $\overrightarrow{\mathcal{T}_\Delta}$. By the definition of $\overrightarrow{\mathcal{T}_\Delta}$, there exist $(q_2, e_0), (q'_2, e'_0) \in \Delta$ such that $\overrightarrow{e_0} = v_0$ and $\overrightarrow{e'_0} = v'_0$, and $q_2 \xrightarrow{u_1|e_1} q_3 \xrightarrow{u_2|e_2} q_3$ and $q'_2 \xrightarrow{u_1|e'_1} q'_3 \xrightarrow{u_2|e'_2} q'_3$ in $\mathcal{T}_\Delta$ such that $\overrightarrow{e_1} = v_1$, $\overrightarrow{e'_1} = v'_1$, $\overrightarrow{e_2} = v_2$, $\overrightarrow{e'_2} = v'_2$.

From $\Delta = \mathsf{extract}_{nc}(g, f, x, \Delta_0, H_3, H_4)$ and H_3H_4 being strongly$-(g,f)-$aligned, we know that there exist $i \geqslant 1$ and two lassos $\xrightarrow{d_0} q_1 \xrightarrow{t_1|d_1} q_2 \xrightarrow{t_2|d_2} q_2$ and $\xrightarrow{d'_0} q'_1 \xrightarrow{t_1|d'_1} q'_2 \xrightarrow{t_2|d'_2} q'_2$ in $\mathcal{T}_{\Delta_0}$, such that $d_2d_1d_0 = e_0g^if$ and $d'_2d'_1d'_0 = e'_0g^if$. Thus we can build two lassos in $\mathcal{T}_{\Delta_0}$: $\xrightarrow{d_0} q_1 \xrightarrow{t_1t_2u_1|e_1d_2d_1} q_3 \xrightarrow{u_2|e_2} q_3$ and $\xrightarrow{d'_0} q'_1 \xrightarrow{t_1t_2u_1|e'_1d'_2d'_1} q'_3 \xrightarrow{u_2|e'_2} q'_3$.

From $\Delta_0 = \mathsf{split}_c(x, H_1, H_2)$ and H_1H_2 being $x-$commuting, we know that there exist $k \geqslant 1$, $c \in \mathcal{C}(B)$ and two lassos $\xrightarrow{c_0} q_0 \xrightarrow{s_1|c_1} q_1 \xrightarrow{s_2|c_2} q_1$ and $\xrightarrow{c'_0} q'_0 \xrightarrow{s_1|c'_1} q'_1 \xrightarrow{s_2|c'_2} q'_1$ in $\mathcal{T}$, such that $c_2c_1c_0[\varepsilon] = d_0c[x^k]$ and $c'_2c'_1c'_0[\varepsilon] = d'_0c[w^k]$. Thus we can build two lassos in $\mathcal{T}$: $\rho_1 : \xrightarrow{c_0} q_0 \xrightarrow{s_1s_2t_1t_2u_1|e_1d_2d_1c_2c_1} q_3 \xrightarrow{u_2|e_2} q_3$ and $\rho_2 : \xrightarrow{c'_0} q'_0 \xrightarrow{s_1s_2t_1t_2u_1|e'_1d'_2d'_1c'_2c'_1} q'_3 \xrightarrow{u_2|e'_2} q'_3$.

By Definition 5.3, for all $j \in \mathbb{N}$, $\mathsf{dist}_f(e_2^j e_1 e_0 g^i f c[x^k], e_2'^j e'_1 e'_0 g^i f c[x^k]) \leqslant L$. As H_1H_2 is non-commuting, by Lemma 5.30, ρ_1 and ρ_2 must also be non-commuting, and thus strongly-balanced. As H_1H_2 and ρ_1 and ρ_2 are strongly-balanced and non-commuting the $g^if c[x^k]$ part can only overlap with itself in the words $e_2^j e_1 e_0 g^i f c[x^k]$ and $e_2'^j e'_1 e'_0 g^i f c[x^k]$. Therefore, we can derive that for all $j \in \mathbb{N}$, $\mathsf{dist}_p(v_0v_1v_2^j, v'_0v'_1v'^j_2) \leqslant L$. □

Proof of Lemma 5.22. By Lemmas 5.31 and 5.32, $\overrightarrow{\mathcal{T}_\Delta}$ and $\overleftarrow{\mathcal{T}_\Delta}$ both satisfy the twinning property. □

5.3.3 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 5.6 (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 $x-\mathrm{commuting}$, for some $x \in B^+$,
   c) or productive, non-commuting and $(f,w)-\mathrm{aligned}$, for some $f \in \mathcal{C}(B)$ and $w \in B^*$.
2. if H_1H_2 is productive and $x-\mathrm{commuting}$, 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 $\mathrm{strongly}-x-\mathrm{commuting}$,
   b) or non-commuting and $\mathrm{strongly}-(h,g,x)-\mathrm{aligned}$, 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 $(f,w)-\mathrm{aligned}$, 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 5.11, 5.21 and 5.22, we have:

Lemma 5.33. *If an S2C $\mathcal{T}$ satisfies the CTP then it satisfies the 2-loop property.*

5.4 Construction of an Equivalent Sequential S2C

Throughout this section, we consider a functional string-to-context transducer $\mathcal{T} = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ 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.

5.4.1 Additional Definitions and Notations

The set of runs of $\mathcal{T}$ is denoted by $\mathcal{R}(\mathcal{T})$. In our construction, we extensively use initial output functions in $\mathcal{F}(Q, \mathcal{C}(B))$ and sets of synchronised runs in $\mathcal{T}^{\leqslant |Q|}$. In this subsection, we define some tools to manipulate those.

We first extend the concatenation and the context-filling operations to initial output functions. Given $\Delta \in \mathcal{F}(Q, \mathcal{C}(B))$, $c \in \mathcal{C}(B)$, $w \in B^*$, we define $\Delta c = \{(q, dc) \mid (q, d) \in \Delta\}$ and $\Delta[w] = \{(q, d[w]) \mid (q, d) \in \Delta\}$.

For a set of synchronised runs $H \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})$, we denote by $\mathsf{word}(H)$ the word read by H. Given $\Delta \in \mathcal{F}(Q, \mathcal{C}(B))$, we define $id_\Delta = (q_i)_{1 \leqslant i \leqslant k} \in \mathcal{R}(\mathcal{T}^k)$, for some enumeration $\{q_1, \ldots, q_k\}$ of $\mathsf{dom}(\Delta)$. As such, $\mathsf{word}(id_\Delta) = \varepsilon$.

We recall the definition of the choose operator, defined in Section 1.1.3, which we will use in this construction. 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 $\mathsf{dom}(\Delta) = \mathsf{dom}(\Delta')$.

Definition 5.7 (Action of $\mathcal{T}$). For $\Delta \subseteq Q \times \mathcal{C}(B)$ and $a \in A$, we define the *action of $\mathcal{T}$ by a on Δ* as $\Delta \bullet a = \mathsf{choose}(\{(q', dc) \mid (q, c) \in \Delta \text{ and } q \xrightarrow[\mathcal{T}]{a|d} q'\})$. Given $H \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})$ and $a \in A$, we define the *action of a on H*, denoted by $H \bullet a$, as the set of runs $H' \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})$ 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 $\mathcal{T}$.

Definition 5.8 (Action of sets of synchronised runs). For $\Delta \subseteq Q \times \mathcal{C}(B)$ and $H \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|})$ a set of runs, we define the *action of $\mathcal{T}$ by H on Δ* as $\Delta \bullet H = \mathsf{choose}(\{(q', dc) \mid (q, c) \in \Delta \text{ and } q \xrightarrow[H]{d} q'\})$.

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.

As $\mathcal{T}$ is functional and is assumed to be trim, if we consider two runs $\xrightarrow{c_1} p_1 \xrightarrow{u|d_1} q$ and $\xrightarrow{c_2} p_2 \xrightarrow{u|d_2} q$, then there exists a run $q \xrightarrow{u|e} f \xrightarrow{g}$ where f is final. By functionality, we have $ged_1c_1[\varepsilon] = ged_2c_2[\varepsilon]$, hence $d_1c_1[\varepsilon] = d_2c_2[\varepsilon]$. This implies that even if the choose operator may select different contexts corresponding to different runs leading to the same state, they yield the same word when they are applied to ε.

Similarly, if we consider two runs $p \xrightarrow{u|d_1} q$ and $p \xrightarrow{u|d_2} q$, and a word w such that $\xrightarrow{c} i \xrightarrow{v|d} p$ with $w = dc[\varepsilon]$, then we have $d_1[w] = d_2[w]$. As a consequence, the choice realised by choose has no impact as soon as one compares the contexts applied to a possible output word produced before.

Last, using a similar reasoning, we can prove that when considering two runs $p_1 \xrightarrow{u|d_1} q$ and $p_2 \xrightarrow{u|d_2} q$ such that p_1 and p_2 appear in some $\Delta : Q \hookrightarrow \mathcal{C}(B)$, obtained after a non-commuting lasso, then we have $d_1\Delta(p_1) = d_2\Delta(p_2)$. Hence, the choice realised by choose has actually no impact.

In the sequel, we will often write equalities involving partial functions $\Delta : Q \hookrightarrow \mathcal{C}(B)$. When these equalities are in one of the three above situations, we will thus omit the operator choose, for simplicity of the writing.

5.4.2 Construction

We now 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:

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

Given a state $\overline{p} = (x, \Delta, H) \in \overline{Q}$, we let $x_{\overline{p}}$ be equal to x. Given a state $\overline{p} = (x, \Delta, H) \in \overline{Q}$ and some run H' in $\mathcal{T}^k$ such that the start state of H' is the end state of H, we let $\overline{p} \bullet H' = (x, \Delta, HH')$.

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 5.3.

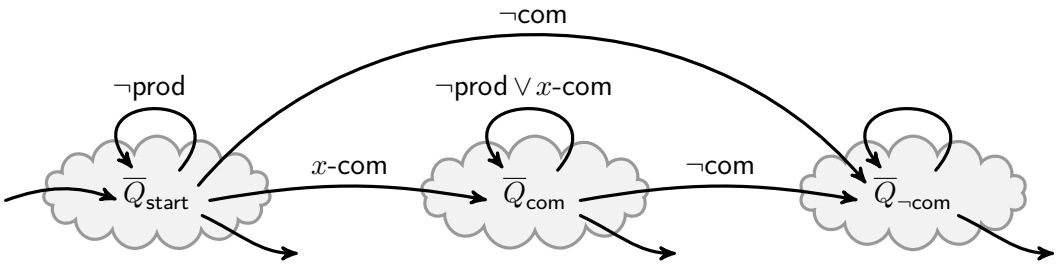


Figure 5.3 – A schematic representation of states and transitions of $\mathcal{D}$.

Initial and final states They are defined as follows:

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

Transitions 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.

For all $\overline{p} \in \overline{Q}$ and $a \in A$, we define the transition $\overline{p} \xrightarrow{a|c} \overline{q}$ where $(\overline{q}, c) = \text{SIMPLIFY}((x, \Delta, H \bullet a))$.

The function SIMPLIFY is performed by Algorithm 5.2, which calls Algorithm 5.1 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. These two algorithms are described below.

We first define EXTEND_WITH_LOOP$(\overline{p}, H_2)$ in Algorithm 5.1 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.

We then define SIMPLIFY$(\overline{p})$ in Algorithm 5.2 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 in the runs stored by the state $\overline{p}$, by using calls to the previous algorithm. Notice that in the case where $x_{\overline{p}} = \bot$, *i.e.* we are in a non-commuting state, we don't call the previous algorithm and just simplify $\overline{p}$ by computing the longest common context of all its stored runs. This can be thought of as applying the longest common prefix strategy of the classical construction for sequential finite-state transducers (cf Chapter 3).

5.4.3 Correctness

In this subsection, we will prove the correctness of our construction. We start with the following easy observation:

Lemma 5.34. *Let $\Delta : Q \hookrightarrow \mathcal{C}(B)$ and H a run in $\mathcal{T}_{\Delta}^{\leqslant |Q|}$. If $H = H_1 H_2$, with $H_1 H_2$ a non-productive lasso. then $\Delta \bullet H_1 = (\Delta \bullet H_1) \bullet H_2$.*

Lemma 5.35 (Correctness of EXTEND_WITH_LOOP)**.** *Let $\overline{p}, \overline{q} \in \overline{Q}_{\infty}$ and $H_2 \in \mathcal{R}(\mathcal{T}_{\Delta_{\overline{p}}}^{\leqslant |Q|})$ such that $x_{\overline{p}} \in B^*$ and $(\overline{q}, c) =$ EXTEND_WITH_LOOP$(\overline{p}, H_2)$.*

— If $x_{\overline{p}} = \varepsilon$ then $(\Delta_{\overline{p}} \bullet H)[\varepsilon] = \Delta_{\overline{q}} c[\varepsilon]$.

— If $x_{\overline{p}} \in B^+$ then for all $k \in \mathbb{N}$, $(\Delta_{\overline{p}} \bullet H)[x_{\overline{p}}^k] = \Delta_{\overline{q}} c[x_{\overline{p}}^k]$.

Proof of Lemma 5.35. We proceed to a case analysis.

Case $x_{\overline{p}} = \varepsilon$ Let $\overline{p} = (\varepsilon, t_{\mathit{init}}, H_1)$. Either H_2 is non-productive, $\overline{q} = \overline{p}$, $c = c_{\varepsilon}$ and, by Lemma 5.34, $\Delta_{\overline{p}} \bullet H_2[\varepsilon] = t_{\mathit{init}} \bullet H_1 H_2[\varepsilon] = t_{\mathit{init}} \bullet H_1[\varepsilon] = \Delta_{\overline{p}}[\varepsilon] = \Delta_{\overline{q}} c[\varepsilon]$.

Otherwise, H_2 is productive and we pass through the **else if** block at Line 4.

Algorithm 5.1 – 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. $H_1 H_2$ is a lasso in $\mathcal{T}_\Delta^{\leqslant |Q|}$.

function EXTEND_WITH_LOOP($\overline{p}, H_2$)
  if H_2 is non-productive **then**
    return $(\overline{p}, c_\varepsilon)$
  else if $\overline{p} = (\varepsilon, t_{init}, H_1)$ **then**
    if $H_1 H_2$ is $x-$commuting, for some $x \in B^+$, **then**
      let $\Delta = \mathsf{split}_c(x, H_1, H_2)$ and $k = \mathsf{pow}_c(x, H_1, H_2)$
      return $((x, \Delta, id_\Delta), (\varepsilon, x^k))$
    else if $H_1 H_2$ is $(f, w)-$aligned, for some $f \in \mathcal{C}(B)$ and $w \in B^*$, **then**
      let $\Delta = \mathsf{split}_{nc}(f, w, H_1, H_2)$
      return $((\bot, \Delta, id_\Delta), f \cdot (\varepsilon, w))$
    end if
  else if $\overline{p} = (x, \Delta_0, H_1)$, where $x \in B^+$, **then**
    if $H_1 H_2$ is strongly$-x-$commuting **then**
      let $k = |\mathsf{out}(H_2)|/|x|$
      return $(\overline{p}, (\varepsilon, x^k))$
    else if $H_1 H_2$ is strongly$-(g, f, x)-$aligned, for some $g, f \in \mathcal{C}(B)$, **then**
      let $\Delta = \mathsf{extract}_{nc}(g, f, x, \Delta_0, H_1, H_2)$
      return $((\bot, \Delta, id_\Delta), gf)$
    end if
  end if
end function

Algorithm 5.2 – Simplifying a state $\overline{p} = (x, \Delta, H) \in \overline{Q}_\infty$.

function SIMPLIFY($\overline{p}$)
  if $\overline{p} = (\bot, \Delta, H)$ **then**
    let $\Delta' = \Delta \bullet H$, $c = \mathsf{lcc}(\Delta')$ and $\overline{q} = (\bot, \Delta'.c^{-1}, id_{\Delta'})$
    return $(\overline{q}, c)$
  else if $\overline{p} = (x, \Delta, H_1 H_2 H_3)$, where $x \in B^*$ and H_2 is the first loop, **then**
    let $\overline{q} = (x, \Delta, H_1)$
    let $(\overline{r}, c) =$ EXTEND_WITH_LOOP$(\overline{q}, H_2)$ with $\overline{r} = (x', \Delta', H')$
    let $(\overline{s}, d) =$ SIMPLIFY$((x', \Delta', H'.H_3))$
    return $(\overline{s}, dc)$
  else
    return $(\overline{p}, c_\varepsilon)$
  end if
end function

If H_1H_2 is $x-$commuting for some $x \in B^+$, then $\Delta = \mathsf{split}_c(x, H_1, H_2)$, $k = \mathsf{pow}_c(x, H_1, H_2)$, $\overline{q} = (x, \Delta, id_\Delta)$ and $c = (\varepsilon, x^k)$. Thus, $\Delta_{\overline{p}} \bullet H_2[\varepsilon] = t_{init} \bullet H_1H_2[\varepsilon] = \Delta[x^k] = \Delta_{\overline{q}}c[\varepsilon]$.

If H_1H_2 is $(f, w)-$aligned for some $f \in \mathcal{C}(B)$ and $w \in B^*$, then we have $\Delta = \mathsf{split}_{nc}(f, w, H_1, H_2)$, $\overline{q} = (\bot, \Delta, id_\Delta)$ and $c = f \cdot (\varepsilon, w)$. Thus, $\Delta_{\overline{p}} \bullet H_2[\varepsilon] = t_{init} \bullet H_1H_2[\varepsilon] = \Delta f[w] = \Delta_{\overline{q}}c[\varepsilon]$.

Case $x_{\overline{p}} \in B^+$ Let $\overline{p} = (x, \Delta_0, H_1)$ and $j \in \mathbb{N}$. Either H_2 is non-productive, $\overline{q} = \overline{p}$, $c = c_\varepsilon$ and, by Lemma 5.34, $\Delta_{\overline{p}} \bullet H_2[x^j] = \Delta_0 \bullet H_1H_2[x^j] = \Delta_0 \bullet H_1[x^j] = \Delta_{\overline{p}}[x^j] = \Delta_{\overline{q}}c[x^j]$.

Otherwise, H_2 is productive and we pass through the **else if** block at Line 12.

If H_1H_2 is strongly$-x-$commuting, then $k = |\mathsf{out}(H_2)|/|x|$, $\overline{q} = \overline{p}$ and $c = (\varepsilon, x^k)$. Thus, $\Delta_{\overline{p}} \bullet H_2[x^j] = \Delta_0 \bullet H_1H_2[x^j] = \Delta_0 \bullet H_1[x^{j+k}] = \Delta_{\overline{q}}c[x^j]$.

If H_1H_2 is strongly$-(g, f, x)-$aligned for some $g, f \in \mathcal{C}(B)$, then $\Delta = \mathsf{extract}_{nc}(g, f, x, \Delta_0, H_1, H_2)$, $\overline{q} = (\bot, \Delta, id_\Delta)$ and $c = gf$. Thus, $\Delta_{\overline{p}} \bullet H_2[x^j] = \Delta_0 \bullet H_1H_2[x^j] = \Delta gf[x^j] = \Delta_{\overline{q}}c[x^j]$. □

In the following lemma, we prove the correctness of the simplification of a state $\overline{p}$. We have three cases. If $x_{\overline{p}} = \varepsilon$ then we have an equality of the output words obtained by filling all the contexts with ε. If $x_{\overline{p}} \in B^+$ then we have an equality of the output words obtained by filling all the contexts with any power of x. If $x_{\overline{p}} = \bot$ then we have an equality of the contexts themselves.

Lemma 5.36 (Correctness of SIMPLIFY). *Let $\overline{p} = (x, \Delta, H) \in \overline{Q}_\infty$ and $(\overline{q}, c) =$ 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 $\Delta_{\overline{p}} = \Delta_{\overline{q}}c$.*

Proof. The result follows from Lemmas 5.37 and 5.38. □

Lemma 5.37 (Correctness of SIMPLIFY for non-commuting states). *Let $\overline{p} \in \overline{Q}_\infty$ such that $x_{\overline{p}} = \bot$ and $(\overline{q}, c) =$ SIMPLIFY$(\overline{p})$. Then $\overline{q} \in \overline{Q}$ and $\Delta_{\overline{p}} = \Delta_{\overline{q}}c$.*

Proof. Let $\overline{p} = (\bot, \Delta, H)$. The fact that $\overline{q} \in \overline{Q}$ is trivial. As $x_{\overline{p}} = \bot$, we only pass through the **if** block at Line 23. Let $\Delta' = \Delta \bullet H$ and $c = \mathsf{lcc}(\Delta')$ and $\overline{q} = (\bot, \Delta' \cdot c^{-1}, id_{\Delta'})$. Thus $\Delta_{\overline{q}}c = (\Delta \bullet H) \cdot c^{-1}c = \Delta_{\overline{p}}$. □

Lemma 5.38 (Correctness of SIMPLIFY for startup and commuting states). *Let $\overline{p} \in \overline{Q}_\infty$ such that $x_{\overline{p}} \neq \bot$ and $(\overline{q}, c) =$ SIMPLIFY$(\overline{p})$. Then $\overline{q} \in \overline{Q}$ and we have:*
— *If $x_{\overline{p}} = \varepsilon$ then $\Delta_{\overline{p}}[\varepsilon] = \Delta_{\overline{q}}c[\varepsilon]$.*
— *If $x_{\overline{p}} \in B^+$ then for all $k \in \mathbb{N}$, $\Delta_{\overline{p}}[x_{\overline{p}}^k] = \Delta_{\overline{q}}c[x_{\overline{p}}^k]$.*

Proof. First observe that the fact that $\overline{q} \in \overline{Q}$ can be proven using a simple induction.

We now consider the second property, and proceed by strong induction on $|H_{\overline{p}}|$. If $H_{\overline{p}} = id$ then, as $x_{\overline{p}} \neq \bot$, we only pass through the **else** statement at Line 31. Then the result is trivially obtained.

Otherwise $|H_{\overline{p}}| > 0$. If $H_{\overline{p}}$ doesn't contain a loop then, again, we only pass through the **else** statement at Line 31, and the result is trivially obtained.

If $H_{\overline{p}}$ contains a loop, we pass through the **else if** block at Line 26. Let $\overline{p} = (x, \Delta, H)$ and $H = H_1H_2H_3$ where H_1H_2 is the first lasso in $H_{\overline{p}}$. Let $\overline{q} = (x, \Delta, H_1)$, $(\overline{r}, c) = \text{EXTEND\_WITH\_LOOP}(\overline{q}, H_2)$, and $(\overline{s}, d) = \text{SIMPLIFY}(\overline{r} \bullet H_3)$. We observe two cases.

Case $x_{\overline{p}} = \varepsilon$ By Lemma 5.35, we have that $t_{\mathit{init}} \bullet H_1 \bullet H_2[\varepsilon] = \Delta_{\overline{q}} \bullet H_2[\varepsilon] = \Delta_{\overline{r}}c[\varepsilon]$.
If $x_{\overline{r}} = \bot$ then, by Lemma 5.37, $\Delta_{\overline{r}} \bullet H_3 = \Delta_{\overline{s}}d$. We obtain that

$$\begin{aligned}
\Delta_{\overline{p}}[\varepsilon] &= t_{\mathit{init}} \bullet H_1H_2H_3[\varepsilon] \\
&= (t_{\mathit{init}} \bullet H_1H_2) \bullet H_3[\varepsilon] \\
&= (\Delta_{\overline{r}}c) \bullet H_3[\varepsilon] \\
&= (\Delta_{\overline{r}} \bullet H_3)c[\varepsilon] \\
&= \Delta_{\overline{s}}dc[\varepsilon]
\end{aligned}$$

If $x_{\overline{r}} \neq \bot$, as $|H_{\overline{r} \bullet H_3}| \leqslant |H_1| + |H_3| < |H_{\overline{p}}|$, then, by the induction hypothesis, Lemma 5.38 holds for $\text{SIMPLIFY}(\overline{r} \bullet H_3)$.
If $x_{\overline{r}} = \varepsilon$, we have that $c = c_\varepsilon$ and $\Delta_{\overline{r}} \bullet H_3[\varepsilon] = \Delta_{\overline{s}}d[\varepsilon]$. We obtain that

$$\begin{aligned}
\Delta_{\overline{p}}[\varepsilon] &= t_{\mathit{init}} \bullet H_1H_2H_3[\varepsilon] \\
&= (t_{\mathit{init}} \bullet H_1H_2) \bullet H_3[\varepsilon] \\
&= (\Delta_{\overline{r}}c) \bullet H_3[\varepsilon] \\
&= \Delta_{\overline{r}} \bullet H_3[\varepsilon] \\
&= \Delta_{\overline{s}}d[\varepsilon] \\
&= \Delta_{\overline{s}}dc[\varepsilon]
\end{aligned}$$

If $x_{\overline{r}} \in B^+$, we have that $c = (\varepsilon, x^\ell)$ for some $\ell \in \mathbb{N}$ and for all $k \in \mathbb{N}$, $\Delta_{\overline{r}} \bullet H_3[x^k] = \Delta_{\overline{s}}d[x^k]$. We obtain that

$$\begin{aligned}
\Delta_{\overline{p}}[\varepsilon] &= t_{\mathit{init}} \bullet H_1H_2H_3[\varepsilon] \\
&= (t_{\mathit{init}} \bullet H_1H_2) \bullet H_3[\varepsilon] \\
&= (\Delta_{\overline{r}}c) \bullet H_3[\varepsilon] \\
&= \Delta_{\overline{r}} \bullet H_3[x^\ell] \\
&= \Delta_{\overline{s}}d[x^\ell] \\
&= \Delta_{\overline{s}}dc[\varepsilon]
\end{aligned}$$

Case $x_{\overline{p}} \in B^+$ By Lemma 5.35, we have that for all $k \in \mathbb{N}$, $t_{init} \bullet H_1 \bullet H_2[x^k] = \Delta_{\overline{q}} \bullet H_2[x^k] = \Delta_{\overline{r}} c[x^k]$. Let $j \in \mathbb{N}$.

If $x_{\overline{r}} = \bot$, then, by Lemma 5.37, $\Delta_{\overline{r}} \bullet H_3 = \Delta_{\overline{s}} d$. We obtain that

$$\begin{aligned}\Delta_{\overline{p}}[x^j] &= t_{init} \bullet H_1 H_2 H_3[x^j] \\ &= (t_{init} \bullet H_1 H_2) \bullet H_3[x^j] \\ &= (\Delta_{\overline{r}} c) \bullet H_3[x^j] \\ &= (\Delta_{\overline{r}} \bullet H_3) c[x^j] \\ &= \Delta_{\overline{s}} dc[x^j]\end{aligned}$$

If $x_{\overline{r}} \neq \bot$, as $|H_{\overline{r} \bullet H_3}| = |H_1| + |H_3| < |H_{\overline{p}}|$, then, by the induction hypothesis, Lemma 5.38 holds for $\textsc{simplify}(\overline{r} \bullet H_3)$.

Also, as $x_{\overline{q}} = x_{\overline{p}} \in B^+$, by construction, it can only happen that $x_{\overline{r}} \in B^+$. Thus we have that $c = (\varepsilon, x^\ell)$ for some $\ell \in \mathbb{N}$ and for all $k \in \mathbb{N}$, $\Delta_{\overline{r}} \bullet H_3[x^k] = \Delta_{\overline{s}} d[x^k]$. We obtain that

$$\begin{aligned}\Delta_{\overline{p}}[x^j] &= t_{init} \bullet H_1 H_2 H_3[x^j] \\ &= (t_{init} \bullet H_1 H_2) \bullet H_3[x^j] \\ &= (\Delta_{\overline{r}} c) \bullet H_3[x^j] \\ &= \Delta_{\overline{r}} \bullet H_3[x^{j+\ell}] \\ &= \Delta_{\overline{s}} d[x^{j+\ell}] \\ &= \Delta_{\overline{s}} dc[x^j]\end{aligned}$$

□

Lemma 5.39 (Correctness of transitions). *For all $\overline{q} \in \overline{Q}$ such that $\overline{i} \xrightarrow[\mathcal{D}]{u|c} \overline{q}$, $\Delta_{\overline{q}} c[\varepsilon] = t_{init} \bullet u[\varepsilon]$.*

Proof. We proceed by induction on $|u|$. If $u = \varepsilon$, the result is obtained trivially. If $u = u'a$ with $a \in A$, let $\overline{p} = (x, \Delta, H) \in \overline{Q}$ such that $\overline{i} \xrightarrow{u'|c} \overline{p} \xrightarrow{a|d} \overline{q}$, and $(\overline{q}, d) = \textsc{simplify}((x, \Delta, H \bullet a))$. By the induction hypothesis, we obtain that $t_{init} \bullet u'[\varepsilon] = \Delta_{\overline{p}} c[\varepsilon]$. By extending with a, we get that $t_{init} \bullet u[\varepsilon] = t_{init} \bullet u' \bullet a[\varepsilon] = \Delta_{\overline{p}} c \bullet a[\varepsilon] = (\Delta_{\overline{p}} \bullet a) c[\varepsilon]$. We observe three cases. If $\overline{p} \in \overline{Q}_{\mathsf{start}}$ then $c = c_\varepsilon$ and, by Lemma 5.38, $(\Delta_{\overline{p}} \bullet a) c[\varepsilon] = \Delta_{\overline{p}} \bullet a[\varepsilon] = \Delta_{\overline{q}} d[\varepsilon] = \Delta_{\overline{q}} dc[\varepsilon]$. If $\overline{p} \in \overline{Q}_{\mathsf{com}}$ then $c[\varepsilon] = x^k$, for some $k \in \mathbb{N}$, and, by Lemma 5.38, $(\Delta_{\overline{p}} \bullet a) c[\varepsilon] = \Delta_{\overline{p}} \bullet a[x^k] = \Delta_{\overline{q}} d[x^k] = \Delta_{\overline{q}} dc[\varepsilon]$. Finally, if $\overline{p} \in \overline{Q}_{\neg\mathsf{com}}$ then, by Lemma 5.37, $(\Delta_{\overline{p}} \bullet a) c[\varepsilon] = \Delta_{\overline{q}} dc[\varepsilon]$. □

5.4.4 Boundedness

We can now prove that all the parts of the built transducer $\mathcal{D}$ are bounded.

Lemma 5.40. *Let* $\overline{p} = (x, \Delta, H_1)$ *and* $\overline{q} = (x', \Delta', H')$, $c \in \mathcal{C}(B)$ *and* $H_2 \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|}_{\Delta \bullet H_1})$, *such that* $(\overline{q}, c) = \text{EXTEND\_WITH\_LOOP}(\overline{p}, H_2)$. *Then* $|word(H')| \leqslant |word(H_1)|$.

Proof. We obtain the result by a trivial case analysis of EXTEND_WITH_LOOP. □

Lemma 5.41. *Let* $\overline{p} = (x, \Delta, H)$ *and* $\overline{s} = (x', \Delta', H')$, *and* $c \in \mathcal{C}(B)$ *such that* $(\overline{s}, c) = \text{SIMPLIFY}(\overline{p})$. *Then* $|word(H')| < |Q|^{|Q|}$.

Proof. We proceed by strong induction on the length of $word(H)$.

If $H = id$ then we can only pass through the **if** block at Line 23 or the **else if** block at Line 31. In both cases, the result is obtained trivially.

Otherwise, let $n = |word(H)|$, and we observe three cases. Firstly, if $x = \bot$, then we pass through the **else if** block at Line 31, and the result is obtained trivially. Secondly, if there is a loop in H, then we pass through the **else if** block at Line 26. Let $H = H_1 H_2 H_3$ where $H_1 H_2$ is the first lasso in $H_{\overline{p}}$. Let $\overline{q} = (x, \Delta, H_1)$, $(\overline{r}, c) = \text{EXTEND\_WITH\_LOOP}(\overline{q}, H_2)$, and $(\overline{s}, d) = \text{SIMPLIFY}(\overline{r} \bullet H_3)$. By Lemma 5.40, we have that $|word(H_{\overline{r}})| \leqslant |word(H_1)|$. Thus $|word(H_{\overline{r} \bullet H_3})| = |word(H_{\overline{r}} H_3)| \leqslant |word(H_1 H_3)| < |word(H)|$, and, by the induction hypothesis applied on $\overline{r} \bullet H_3$, we obtain the result. Thirdly, if there is no loop in H, then we pass through the **else if** block at Line 31 and we have $|word(H')| = |word(H)| < |Q|^{|Q|}$. Indeed, suppose we had that $|word(H)| \geqslant |Q|^{|Q|}$, then H must contain a loop, which is a contradiction. □

Lemma 5.42. *Let* $\overline{p} = (x, \Delta, H_1)$, $\overline{q} = (x', \Delta', H')$, $c \in \mathcal{C}(B)$ *and* $H_2 \in \mathcal{R}(\mathcal{T}^{\leqslant |Q|}_{\Delta \bullet H_1})$ *such that* $(\overline{q}, c) = \text{EXTEND\_WITH\_LOOP}(\overline{p}, H_2)$. *We assume that* $|word(H_1 H_2)| \leqslant |Q|^{|Q|}$, $|x| \leqslant M_{\mathcal{T}} |Q|^{|Q|}$ *and for all* $(q, d) \in \Delta$, $|d| \leqslant M_{\mathcal{T}} |Q|^{|Q|}$. *Then we distinguish two cases:*

— *if* $x' \neq \bot$, *then* $|x'| \leqslant M_{\mathcal{T}} |Q|^{|Q|}$ *and for all* $(q, d) \in \Delta'$, $|d| \leqslant M_{\mathcal{T}} |Q|^{|Q|}$,

— *if* $x' = \bot$, *then for all* $(q, d) \in \Delta'$, $|d| \leqslant 2 M_{\mathcal{T}} |Q|^{|Q|}$.

Proof. We proceed by case analysis of EXTEND_WITH_LOOP. If H_2 is not productive then we pass through the **if** block at Line 2, the returned state is $\overline{p}$ and the result is obtained trivially. Otherwise, H_2 is productive.

If $x = \varepsilon$ then we pass through the **else if** block at Line 4. Either $H_1 H_2$ is $x'-\text{commuting}$, for some $x' \in B^+$ such that $|x'| \leqslant M_{\mathcal{T}} |Q|^{|Q|}$, and we let $\Delta' = \mathsf{split}_c(x', H_1, H_2)$, or $H_1 H_2$ is $(f, w)-\text{aligned}$, for some $f \in \mathcal{C}(B)$ and $w \in B^*$, and we let $\Delta' = \mathsf{split}_{nc}(f, w, H_1, H_2)$. In both cases, by definition of Δ', we have that, for all $(q, d) \in \Delta'$, $|d| \leqslant M_{\mathcal{T}} + |\mathsf{out}(H_1)| \leqslant M_{\mathcal{T}} |Q|^{|Q|}$, because $|\mathsf{word}(H_2)| \geqslant 1$ and $|word(H_1 H_2)| \leqslant |Q|^{|Q|}$.

If $x \in B^+$ then we pass through the **else if** block at Line 12. If $H_1 H_2$ is $\text{strongly}-x-\text{commuting}$, then the returned state is $\overline{p}$ and the result is obtained trivially. If $H_1 H_2$ is $\text{strongly}-(g, f, x)-\text{aligned}$, for some $f, g \in \mathcal{C}(B)$, then we let $\Delta' = \mathsf{extract}_{nc}(g, f, x, \Delta, H_1, H_2)$. By definition of Δ', we have that, for all $(q, d) \in \Delta'$, $|d| \leqslant |\Delta(q)| + |\mathsf{out}(H_1)| \leqslant 2 M_{\mathcal{T}} |Q|^{|Q|}$, because $|\Delta(q)| \leqslant M_{\mathcal{T}} |Q|^{|Q|}$ and $|word(H_1)| \leqslant |Q|^{|Q|}$. □

Lemma 5.43. *Let $\overline{p} = (x, \Delta, H)$ and $\overline{s} = (x', \Delta', H')$, and $c \in \mathcal{C}(B)$ such that $(\overline{s}, c) = \text{SIMPLIFY}(\overline{p})$ and $x, x' \in B^*$. If $|word(H)| \leqslant |Q|^{|Q|}$, $|x| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$ and for all $(q, d) \in \Delta$, $|d| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$ then $|x'| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$ and for all $(q, d) \in \Delta'$, $|d| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$.*

Proof. We proceed by strong induction on the length of $word(H)$. Let $n = |word(H)|$, we observe two cases. First, if there is a loop in H, then we pass through the **else if** block at Line 26. Let $H = H_1H_2H_3$ where H_1H_2 is the first lasso in $H_{\overline{p}}$. Let $\overline{q} = (x, \Delta, H_1)$, $(\overline{r}, c) = \text{EXTEND\_WITH\_LOOP}(\overline{q}, H_2)$, $\overline{r} = (x'', \Delta'', H'')$, and $(\overline{s}, d) = \text{SIMPLIFY}(\overline{r} \bullet H_3)$. We have that $|word(H_1H_2)| \leqslant |word(H)| \leqslant |Q|^{|Q|}$. Then, by Lemma 5.42, we have that $|x''| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$ and for all $(q, d) \in \Delta''$, $|d| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$. Again, by Lemma 5.40, we have that $|word(H_{\overline{r}})| \leqslant |word(H_1)|$. Thus $|word(H_{\overline{r}\bullet H_3})| = |word(H_{\overline{r}}H_3)| \leqslant |word(H_1H_3)| < |word(H)| \leqslant |Q|^{|Q|}$, and, by the induction hypothesis applied on $\overline{r} \bullet H_3$, we obtain the result. Second, if there is no loop in H, then we pass through the **else if** block at Line 31 and the result is obtained trivially. □

Lemma 5.44 (Boundedness of $\mathcal{D}$). *For all $\overline{q} = (x', \Delta', H') \in \overline{Q}$ such that $\overline{i} \xrightarrow[\mathcal{D}]{u|c} \overline{q}$, the following assertions are satisfied:*

- *$|x'| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$,*
- *$|word(H')| < |Q|^{|Q|}$,*
- *if $x' \neq \bot$, then for all $(q, d) \in \Delta'$, $|d| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$,*
- *if $x' = \bot$, then for all $(q, d) \in \Delta'$, $|d| \leqslant 4M_{\mathcal{T}}|Q|^{|Q|+2}$.*

Proof. We distinguish two cases, whether $x' = \bot$ or not.

We start with the case $x' \neq \bot$ and proceed by induction on $|u|$. If $u = \varepsilon$, the result is obtained trivially. If $u = u'a$ with $a \in A$, let $\overline{p} = (x, \Delta, H) \in \overline{Q}$ such that $\overline{i} \xrightarrow{u'|c} \overline{p} \xrightarrow{a|d} \overline{q}$, and $(\overline{q}, d) = \text{SIMPLIFY}((x, \Delta, H \bullet a))$. By the induction hypothesis, we have that $|x| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$, for all $(q, d) \in \Delta$, $|d| \leqslant M_{\mathcal{T}}|Q|^{|Q|}$, and $|word(H)| < |Q|^{|Q|}$. By extending with a, we have that $|word(H \bullet a)| \leqslant |Q|^{|Q|}$. Then by Lemma 5.43, we obtain the result.

We now consider that $x' = \bot$. The execution $\overline{i} \xrightarrow[\mathcal{D}]{u|c} \overline{q}$ can be decomposed as $\overline{i} \xrightarrow[\mathcal{D}]{u_1|c_1} \overline{p} \xrightarrow[\mathcal{D}]{a|c_2} \overline{p}' \xrightarrow[\mathcal{D}]{u_2|c_3} \overline{q}$, with $x_{\overline{p}} \neq \bot$ and $x_{\overline{p}'} = \bot$. The transition from $\overline{p}$ to $\overline{p}'$ involves the removal of a loop, by EXTEND_WITH_LOOP, which is non-commuting. As a consequence of the 2-loop property, we have that some intermediate state $\overline{p}'' = (\bot, \Delta, id_\Delta)$ is computed, and that $\overleftarrow{\mathcal{T}_\Delta}$ and $\overrightarrow{\mathcal{T}_\Delta}$ both satisfy the classical twinning property. In addition, thanks to the first case of this proof, and to Lemma 5.42, we also have that for all $(q, d) \in \Delta'$, $|d| \leqslant 2M_{\mathcal{T}}|Q|^{|Q|}$. The behaviour of our procedure starting from this intermediate state $\overline{p}''$ is exactly the one of the determinisation procedure of Choffrut (cf Chapter 3) performed on the two sides of the context. See for details Line 24 of Algorithm 5.2. By Lemma 3.3, we know that delays stored in the determinisation procedure of

Choffrut have size at most $2M(n^2+1)$, where M is the size of the largest output of the transducer, and n is the number of states. We therefore consider the left and right transducers from Δ', each having $|Q|$ states and using $M = 2M_{\mathcal{T}}|Q|^{|Q|}$. As a consequence, we obtain that for all $(q,d) \in \Delta'$, we have :

$$|d| \leqslant 2.2.(2M_{\mathcal{T}}|Q|^{|Q|}).(|Q|^2+1) \leqslant 16M_{\mathcal{T}}|Q|^{|Q|+2} \qquad \square$$

5.4.5 Final Theorem

Theorem 5.45. *$\mathcal{D}$ is a finite sequential string-to-context transducer equivalent to $\mathcal{T}$.*

Proof. The fact that $\mathcal{D}$ is deterministic is direct by an observation of its definition. In addition, $\mathcal{D}$ is finite as a consequence of Lemma 5.44. To prove the equivalence between $\mathcal{D}$ and $\mathcal{T}$, we consider a word $u \in A^*$, and the run $\bar{i} \xrightarrow[\mathcal{D}]{u|c} \bar{q}$ of $\mathcal{D}$ on u. By Lemma 5.39, we have $\Delta_{\bar{q}}c[\varepsilon] = t_{\mathit{init}} \bullet u[\varepsilon]$. This entails:

$$\begin{aligned} u \in \mathsf{dom}([\![\mathcal{T}]\!]) &\iff \mathsf{dom}(t_{\mathit{init}} \bullet u) \cap \mathsf{dom}(t_{\mathit{final}}) \neq \emptyset \\ &\iff \mathsf{dom}(\Delta_{\bar{q}}) \cap \mathsf{dom}(t_{\mathit{final}}) \neq \emptyset \\ &\iff \bar{q} \in \mathsf{dom}(\overline{t_{\mathit{final}}}) \\ &\iff u \in \mathsf{dom}([\![\mathcal{D}]\!]) \end{aligned}$$

The definition of $\overline{t_{\mathit{final}}}$ then directly implies $[\![T]\!] = [\![\mathcal{D}]\!]$. $\square$

5.5 Deciding Sequentiality of S2Cs

In this section, we prove the following result:

Theorem 5.46. *Given a functional string-to-context transducer, deciding 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 5.9 (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 5.6).

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

Lemma 5.48. *If a string-to-context transducer $\mathcal{T}$ satisfies the small-2-loop property then the* SIMPLIFY *procedure is well-defined for arbitrarily-long pairs of runs in $\mathcal{T}^2$.*

Proof. From an arbitrary pair of runs in $\mathcal{T}^2$, it is always possible to find lassos for which both the initial part and the loop part have lengths less than $|Q|^2$. □

We can then prove that if a string-to-context transducer satisfies the small-2-loop property then the function it realizes satisfies the contextual Lipschitz property.

Lemma 5.49. *If a string-to-context transducer $\mathcal{T}$ satisfies the small-2-loop property then for all runs $\rho_1 :\xrightarrow{c_1} i_1 \xrightarrow{u|d_1} q_1$ and $\rho_2 :\xrightarrow{c_2} i_2 \xrightarrow{u|d_2} q_2$, with i_1, i_2 initial states, we have $\mathsf{dist}_f(d_1c_1[\varepsilon], d_2c_2[\varepsilon]) \leqslant 34M_{\mathcal{T}}|Q|^{|Q|+2}$.*

Proof. Let H be the set of runs containing only the two runs ρ_1 and ρ_2, and let $(\overline{q}, c) = \text{SIMPLIFY}(\bar{i} \bullet H)$. By Lemma 5.38 and Lemma 5.48, we have that $t_{init} \bullet H[\varepsilon] = \Delta_{\bar{i}\bullet H}[\varepsilon] = \Delta_{\overline{q}}c[\varepsilon]$. Let $e_1, e_2 \in \mathcal{C}(B)$ are such that $\Delta_{\overline{q}} = \{(q_1, e_1), (q_2, e_2)\}$. We thus have $d_1c_1[\varepsilon] = e_1c[\varepsilon]$ and $d_2c_2[\varepsilon] = e_2c[\varepsilon]$. Therefore $\mathsf{dist}_f(d_1c_1[\varepsilon], d_2c_2[\varepsilon]) \leqslant |e_1| + |e_2|$. Let $\overline{q} = (x_{\overline{q}}, \Delta, H_{\overline{q}})$. By definition, we know that $\Delta_{\overline{q}} = \Delta \bullet H_{\overline{q}}$ and that, by Lemma 5.44, for all $(q, d) \in \Delta$, $|d| \leqslant 4M_{\mathcal{T}}|Q|^{|Q|+2}$, and $|word(H_{\overline{q}})| < |Q|^{|Q|}$. Thus $|e_1| \leqslant 17M_{\mathcal{T}}|Q|^{|Q|+2}$ and $|e_2| \leqslant 17M_{\mathcal{T}}|Q|^{|Q|+2}$ and we obtain the result. □

Proof of Lemma 5.47. Let $\mathcal{T}$ be a string-to-context transducer. Assume that $\mathcal{T}$ satisfies the small-2-loop property and let $u_1, u_2 \in \mathsf{dom}(\llbracket\mathcal{T}\rrbracket)$. We want to prove that there exists $K \in \mathbb{N}$ such that $\mathsf{dist}_f(\llbracket\mathcal{T}\rrbracket(u_1), \llbracket\mathcal{T}\rrbracket(u_2)) \leqslant K\mathsf{dist}_p(u_1, u_2)$. If $u_1 = u_2$ then $\mathsf{dist}_f(\llbracket\mathcal{T}\rrbracket(u_1), \llbracket\mathcal{T}\rrbracket(u_2)) = 0$, and the result is trivially obtained, whatever the value of K is.

In the following, we assume that $u_1 \neq u_2$ and thus $\mathsf{dist}_p(u_1, u_2) \geqslant 1$. Let $\rho_1 :\xrightarrow{c_1} i_1 \xrightarrow{u_1|d_1} f_1 \xrightarrow{e_1}$ and $\rho_2 :\xrightarrow{c_2} i_2 \xrightarrow{u_2|d_2} f_2 \xrightarrow{e_2}$ be the corresponding runs in $\mathcal{T}$. Let $u = \mathsf{lcp}(u_1, u_2)$ and u_1', u_2' such that $u_1 = uu_1'$ and $u_2 = uu_2'$. Let $p_1, p_2 \in Q$ the states that ρ_1 and ρ_2 reach after having read u. That is $\rho_1 :\xrightarrow{c_1} i_1 \xrightarrow{u|d_1'} p_1 \xrightarrow{u_1'|d_1''} f_1 \xrightarrow{e_1}$ and $\rho_2 :\xrightarrow{c_2} i_2 \xrightarrow{u|d_2'} p_2 \xrightarrow{u_2'|d_2''} f_2 \xrightarrow{e_2}$.

By Lemma 5.49, $\mathsf{dist}_f(d_1'c_1[\varepsilon], d_2'c_2[\varepsilon]) \leqslant 34M_{\mathcal{T}}|Q|^{|Q|+2}$. Therefore,

$$\begin{aligned}
\mathsf{dist}_f(\llbracket\mathcal{T}\rrbracket(u_1), \llbracket\mathcal{T}\rrbracket(u_2)) &= \mathsf{dist}_f(e_1d_1c_1[\varepsilon], e_2d_2c_2[\varepsilon]) \\
&\leqslant 34M_{\mathcal{T}}|Q|^{|Q|+2} + |e_1d_1''| + |e_2d_2''| \\
&\leqslant 34M_{\mathcal{T}}|Q|^{|Q|+2} + M_{\mathcal{T}}(|u_1'| + 1) + M_{\mathcal{T}}(|u_2'| + 1) \\
&\leqslant M_{\mathcal{T}}(34|Q|^{|Q|+2} + \mathsf{dist}_p(u_1, u_2) + 2) \\
&\leqslant M_{\mathcal{T}}(34|Q|^{|Q|+2} + 3)\mathsf{dist}_p(u_1, u_2)
\end{aligned}$$

□

Proof of Theorem 5.46. By Theorem 5.8 and Lemma 5.47, $\mathcal{T}$ admits an equivalent sequential S2C transducer iff $\mathcal{T}$ satisfies the small-2-loop property. Because

of this equivalence, we give 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 one of the following four shapes:

1. a run $H : \xrightarrow{(c_0,d_0)} (i_1,i_2) \xrightarrow{u_1|(c_1,d_1)} (p_1,p_2) \xrightarrow{u_2|(c_2,d_2)} (p_1,p_2)$ in $\mathcal{T}^2$, with $|u_1| < |Q|^2$ and $|u_2| < |Q|^2$, that is a productive lasso neither commuting nor aligned.
2. a run $H : \xrightarrow{(c_0,d_0)} (i_1,i_2) \xrightarrow{u_1|(c_1,d_1)} (p_1,p_2) \xrightarrow{u_2|(c_2,d_2)} (p_1,p_2) \xrightarrow{u_3|(c_3,d_3)} (q_1,q_2) \xrightarrow{u_4|(c_4,d_4)} (q_1,q_2)$ in $\mathcal{T}^2$, with $|u_i| < |Q|^2$, for all $i \in \{1,\dots,4\}$, such that the first lasso is a productive $x-\text{commuting}$ lasso, for some $x \in B^+$, and the second lasso is a productive lasso neither $\text{strongly}-x-\text{commuting}$ nor strongly aligned.
3. a run $H : \xrightarrow{(c_0,d_0)} (i_1,i_2) \xrightarrow{u_1|(c_1,d_1)} (p_1,p_2) \xrightarrow{u_2|(c_2,d_2)} (p_1,p_2)$ in $\mathcal{T}^2$, with $|u_1| < |Q|^2$ and $|u_2| < |Q|^2$, that is a productive aligned lasso but, for Δ appropriately obtained with split_{nc}, $\overleftarrow{\mathcal{T}_\Delta}$ and/or $\overrightarrow{\mathcal{T}_\Delta}$ do not satisfy the twinning property.
4. a run $H : \xrightarrow{(c_0,d_0)} (i_1,i_2) \xrightarrow{u_1|(c_1,d_1)} (p_1,p_2) \xrightarrow{u_2|(c_2,d_2)} (p_1,p_2) \xrightarrow{u_3|(c_3,d_3)} (q_1,q_2) \xrightarrow{u_4|(c_4,d_4)} (q_1,q_2)$ in $\mathcal{T}^2$, with $|u_i| < |Q|^2$, for all $i \in \{1,\dots,4\}$, such that the first lasso is a productive aligned lasso, the second lasso is productive and strongly aligned but, for Δ appropriately obtained with $\mathsf{extract}_{nc}$, $\overleftarrow{\mathcal{T}_\Delta}$ and/or $\overrightarrow{\mathcal{T}_\Delta}$ do not satisfy the twinning property.

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 $x-\text{commuting}$ (resp. no $f \in \mathcal{C}(B)$ and $w \in B^*$ such that the lasso is $(f,w)-\text{aligned}$). Verifying that a lasso in $\mathcal{T}^2$ is not aligned boils down to checking whether there exists no $f \in \mathcal{C}(B)$ and $w \in B^*$ such that the lasso is $(f,w)-\text{aligned}$. 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 the verification for shape 1 can be done in polynomial time. Similarly, the verification for shapes 2, 3 and 4 can be done in polynomial time. Furthermore, all three shapes are of polynomial size, by definition of the small-2-loop property, yielding the result. The existence of an equivalent sequential S2C is therefore in coNP. □

5.6 Related Work

In this chapter, we presented a characterisation of the functional string-to-context transducers that admit a sequential equivalent. We have seen that,

thanks to the equivalence between functional string-to-context transducers and copyful concatenation-free deterministic streaming string transducers (*cf.* Chapter 2), this is a first step towards solving the problem of the register minimisation for copyful concatenation-free deterministic streaming string transducers.

The authors of [Bas+16] solved the register minimisation problem in a similar setting: the copyless concatenation-free non-deterministic streaming string transducers. Their result relies on the use of sweeping transducers, which are two-way finite-state transducers that are only allowed to change the direction of the reading head at the extremities of the input word.

They first prove that copyless concatenation-free non-deterministic streaming string transducers with k registers are equivalent to (non-deterministic) functional sweeping transducers that execute at most $2k$ passes over the input word. They then prove that it is decidable, given a natural number k, whether a functional sweeping transducer admits an equivalent sweeping transducer that executes at most k passes, and devise a construction for this equivalent transducer if it exists.

It should be stressed that the problem they study is quite different from ours, as they focus on copyless non-deterministic streaming string transducers whereas we target the copyful deterministic streaming string transducers.

Conclusion

In this manuscript, we studied several problems around the simplification of transducers in order to make their evaluation or translation into programs more efficient. We have seen that these problems can be solved with tools that stem from the work of [Cho77] around the sequentiality of finite-state transducers.

In Chapter 4, we have presented a characterisation of the functional finite-state transducers that admit a k-sequential equivalent, for some $k \in \mathbb{N}$. This characterisation was based on a property of the functions realised by the transducers, the Lipschitz property of order k, and a property of the transducers themselves, the branching twinning property of order k. We devised a decision procedure for the k-sequentiality problem, a construction for the k-sequential equivalent, if it exists, and discussed how we can search for a minimal such k.

As a by-product of our work, we were able to solve the problem of the minimisation to k registers of copyless appending deterministic streaming string transducers. Prior to this work, [DRT16] had devised another generalisation of the results of [Cho77] to tackle the problem of the minimisation to k registers of copyful appending deterministic streaming string transducers. It is interesting to notice that their characterisation involved a generalisation of the bounded variation property whereas our work involved a generalisation of the Lipschitz property. For both these generalisations, the case for $k = 1$ coincides with the sequentiality problem of functional finite-state transducers. We have discussed the fact that, for $k > 1$, the branching twinning property of order k is stronger than the twinning property of order k of [DRT16]. Henceforth, we obtain that, for $k > 1$, the Lipschitz property of order k is also a stronger property than the bounded variation property of order k, the primmer characterising the copyless appending deterministic streaming string transducers with k registers and the latter characterising the copyful appending deterministic streaming string transducers with k registers.

In Chapter 5, we have presented a characterisation of the functional string-to-context transducers that admit a sequential equivalent. This characterisation used two properties of the functions realised by the transducers, the contextual bounded variation property and the contextual Lipschitz property, and a property of the transducers themselves, the contextual twinning property. We also devised a decision procedure for this problem, along with a construction for the sequential equivalent, if it exists.

There are still some work to do to complete the picture. First, we have not yet

proved a lower bound for the decision procedure for the sequentiality problem of string-to-context transducers. Second, it would be interesting to see if and how we can adapt our procedure to decide the problem of the functionality of string-to-context transducers, without relying on the functionality of non-deterministic streaming string transducers.

We conclude with some broader perspectives. We were able to characterise the functions realised by sequential string-to-context transducers with both a variant of the bounded variation property and a variant of the Lipschitz property. We thus expect that this work will be generalisable further to solve both the register minimisation of copyful concatenation-free deterministic streaming string transducers, in the vein of [DRT16], and the register minimisation of copyless concatenation-free deterministic streaming string transducers, similarly to our work on k-sequentiality.

List of Figures

List of Tables

Bibliography

[AHU69] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. “A General Theory of Translation”. en. In: *Mathematical systems theory* 3.3 (Sept. 1969), pp. 193–221 (cit. on p. 14).

[AČ10] Rajeev Alur and Pavol Černý. “Expressiveness of Streaming String Transducers”. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*. 2010, pp. 1–12 (cit. on pp. iii, v, 4, 15, 17, 25).

[AČ11] Rajeev Alur and Pavol Černý. “Streaming Transducers for Algorithmic Verification of Single-Pass List-Processing Programs”. In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’11. New York, NY, USA: ACM, 2011, pp. 599–610 (cit. on pp. iii, v, 4, 15, 19, 21, 23).

[Alu+13] Rajeev Alur, Loris D’Antoni, Jyotirmoy Deshmukh, et al. “Regular Functions and Cost Register Automata”. In: *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 13–22 (cit. on pp. 4, 26).

[AD11] Rajeev Alur and Jyotirmoy V. Deshmukh. “Nondeterministic Streaming String Transducers”. In: *International Colloquium on Automata, Languages, and Programming*. Springer, 2011, pp. 1–20 (cit. on pp. 21–23).

[AR13] Rajeev Alur and Mukund Raghothaman. “Decision Problems for Additive Regular Functions”. In: *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*. Vol. 7966. Lecture Notes in Computer Science. Springer, 2013, pp. 37–48 (cit. on p. 67).

[Ari96] *ARIANE 5 - Flight 501 Failure*. Tech. rep. by the Inquiry Board. Chairman : J-L Lions. Paris, 1996 (cit. on p. 1).

[Bas+16] Félix Baschenis, Olivier Gauwin, Anca Muscholl, et al. “Minimizing Resources of Sweeping and Streaming String Transducers”. In: *43rd International Colloquium on Automata, Languages, and Pro-*

gramming, ICALP 2016, July 11-15, 2016, Rome, Italy. 2016, 114:1–114:14 (cit. on p. 103).

[Bas+18] Félix Baschenis, Olivier Gauwin, Anca Muscholl, et al. "One-Way Definability of Two-Way Word Transducers". en. In: *Logical Methods in Computer Science ; Volume 14* (2018), Issue 4, 1860–5974 (cit. on p. 36).

[BC02] Marie-Pierre Béal and Olivier Carton. "Determinization of Transducers over Finite and Infinite Words". In: *Theoretical Computer Science* 289.1 (Oct. 2002), pp. 225–251 (cit. on pp. 4, 13, 39, 48).

[Béa+00] Marie-Pierre Béal, Olivier Carton, Christophe Prieur, et al. "Squaring Transducers: An Efficient Procedure for Deciding Functionality and Sequentiality of Transducers". In: *LATIN 2000: Theoretical Informatics*. Springer, 2000, pp. 397–406 (cit. on pp. 4, 13, 48).

[Ber13] Jean Berstel. *Transductions and Context-Free Languages*. Springer-Verlag, 2013 (cit. on pp. 3, 12, 26, 41).

[BH77] Meera Blattner and Tom Head. "Single-Valued a-Transducers". In: *Journal of Computer and System Sciences* 15.3 (Dec. 1977), pp. 310–327 (cit. on pp. 13, 23).

[BH79] Meera Blattner and Tom Head. "The Decidability of Equivalence for Deterministic Finite Transducers". In: *Journal of Computer and System Sciences* 19.1 (1979), pp. 45–49 (cit. on pp. 13, 23).

[CL11] Arnaud Carayol and Christof Löding. "Uniformization in Automata Theory". en. In: *14th Congress of Logic, Methodology and Philosophy of Science*. College Publications, July 2011, pp. 153–178 (cit. on p. 2).

[CS86] C. Choffrut and M. P. Schutzenberger. "Décomposition de Fonctions Rationnelles". fr. In: *STACS 86*. Ed. by B. Monien and G. Vidal-Naquet. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1986, pp. 213–226 (cit. on pp. 5, 51).

[Cho77] Christian Choffrut. "Une Caracterisation Des Fonctions Sequentielles et Des Fonctions Sous-Sequentielles En Tant Que Relations Rationnelles". In: *Theoretical Computer Science* 5.3 (Dec. 1977), pp. 325–337 (cit. on pp. iii, v, 4, 6, 13, 36, 39–41, 44, 48, 52, 70, 73, 105).

[Cla+18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, et al., eds. *Handbook of Model Checking*. en. Springer International Publishing, 2018 (cit. on p. 2).

[Com+07] H. Comon, M. Dauchet, R. Gilleron, et al. *Tree Automata Techniques and Applications*. Published: Available on: http://www.grappa.univ-lille3.fr/tata. 2007 (cit. on p. 17).

[CK87] Karel Culik II and Juhani Karhumäki. “The Equivalence Problem for Single-Valued Two-Way Transducers (on NPDTOL Languages) Is Decidable”. In: *SIAM J. Comput.* 16.2 (Apr. 1987), pp. 221–230 (cit. on pp. 15, 23).

[Dav+17] Laure Daviaud, Ismaël Jecker, Pierre-Alain Reynier, et al. “Degree of Sequentiality of Weighted Automata”. en. In: *Foundations of Software Science and Computation Structures*. Ed. by Javier Esparza and Andrzej S. Murawski. Vol. 10203. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 215–230 (cit. on pp. 5, 36, 51).

[DRT16] Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. “A Generalised Twinning Property for Minimisation of Cost Register Automata”. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS ’16, New York, NY, USA, July 5-8, 2016*. 2016, pp. 857–866 (cit. on pp. 4, 6, 50, 52–54, 70, 105, 106).

[EH01] Joost Engelfriet and Hendrik J. Hoogeboom. “MSO Definable String Transductions and Two-Way Finite-State Transducers”. In: *ACM Trans. Comput. Logic* 2.2 (Apr. 2001), pp. 216–254 (cit. on pp. 14, 15, 23, 25).

[EFJ18] Léo Exibard, Emmanuel Filiot, and Ismaël Jecker. “The Complexity of Transducer Synthesis from Multi-Sequential Specifications”. In: *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Ed. by Igor Potapov, Paul Spirakis, and James Worrell. Vol. 117. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 46:1–46:16 (cit. on p. 50).

[FGR15] Emmanuel Filiot, Raffaella Gentilini, and Jean-FranÃ§ois Raskin. “Quantitative Languages Defined by Functional Automata”. In: *Logical Methods in Computer Science* 11.3 (Sept. 2015), p. 14. arXiv: 1111.0862 (cit. on p. 50).

[FMR18] Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. “A Pattern Logic for Automata with Outputs”. In: *Developments in Language Theory*. Ed. by Mizuho Hoshi and Shinnosuke Seki. Vol. 11088. Cham: Springer International Publishing, 2018, pp. 304–317 (cit. on pp. 13, 23, 48).

[FR17] Emmanuel Filiot and Pierre-Alain Reynier. “Copyful Streaming String Transducers”. en. In: *Reachability Problems*. Ed. by Matthew Hague and Igor Potapov. Vol. 10506. Cham: Springer International Publishing, 2017, pp. 75–86 (cit. on pp. 19, 23).

[FW65] Nathan J. Fine and Herbert S. Wilf. “Uniqueness Theorems for Periodic Functions”. In: *Proceedings of the American Mathematical Society* 16 (1965), pp. 109–114 (cit. on p. 10).

[Gur80] Eitan M. Gurari. “The Equivalence Problem for Deterministic Two-Way Sequential Transducers Is Decidable”. In: *Siam Journal on Computing - SIAMCOMP*. Vol. 11. Nov. 1980, pp. 83–85 (cit. on pp. 15, 23).

[GI83] Eitan M. Gurari and Oscar H. Ibarra. “A Note on Finite-Valued and Finitely Ambiguous Transducers”. In: *Mathematical systems theory* 16 (1983), pp. 61–66 (cit. on p. 13).

[Iba77] Oscar H. Ibarra. “The Unsolvability of the Equivalence Problem for E-Free NGSM’s with Unary Input (Output) Alphabet and Applications”. In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. SFCS ’77. Washington, DC, USA: IEEE Computer Society, 1977, pp. 74–81 (cit. on pp. 13, 15, 22, 23).

[JF15] Ismaël Jecker and Emmanuel Filiot. “Multi-Sequential Word Relations”. en. In: *Developments in Language Theory*. Ed. by Igor Potapov. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 288–299 (cit. on pp. 5, 50, 51).

[Kir12] Daniel Kirsten. “Decidability, Undecidability, and PSPACE-Completeness of the Twins Property in the Tropical Semiring”. In: *Theoretical Computer Science* 420 (Feb. 2012), pp. 56–63 (cit. on p. 50).

[KL09] Daniel Kirsten and Sylvain Lombardy. “Deciding Unambiguity and Sequentiality of Polynomially Ambiguous Min-Plus Automata”. In: *STACS*. 2009 (cit. on p. 50).

[LT93] N. G. Leveson and C. S. Turner. “An Investigation of the Therac-25 Accidents”. In: *IEEE Computer* 26.7 (July 1993), pp. 18–41 (cit. on p. 1).

[LS06] Sylvain Lombardy and Jacques Sakarovitch. “Sequential?” In: *Theoretical Computer Science*. In Honour of Professor Christian Choffrut on the Occasion of His 60th Birthday 356.1 (May 2006), pp. 224–244 (cit. on p. 50).

[Lot02] M. Lothaire. *Algebraic Combinatorics on Words*. Encyclopedia of Mathematics and Its Applications. Cambridge University Press, 2002 (cit. on p. 10).

[MP19] Anca Muscholl and Gabriele Puppis. “The Many Facets of String Transducers”. In: 2019 (cit. on pp. 15, 22).

[Pnu77] A. Pnueli. “The Temporal Logic of Programs”. In: *18th Annual Symposium on Foundations of Computer Science (Sfcs 1977)*. Oct. 1977, pp. 46–57 (cit. on p. 2).

[RV19] Pierre-Alain Reynier and Didier Villevalois. “Sequentiality of String-to-Context Transducers”. In: *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Ed. by Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, et al. Vol. 132. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 128:1–128:14 (cit. on pp. 21, 23, 36, 69).

[Saa15] Aleksi Saarela. “Systems of Word Equations, Polynomials and Linear Algebra: A New Approach”. In: *European Journal of Combinatorics* 47 (2015), pp. 1–14 (cit. on p. 71).

[Sak09] Jacques Sakarovitch. *Elements of Automata Theory*. Ed. by Reuben-Translator Thomas. Cambridge University Press, 2009 (cit. on pp. 3, 12).

[Sch75] Marcel P. Schutzenberger. “Sur les relations rationnelles”. fr. In: *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20–23, 1975*. Ed. by H. Brakhage. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1975, pp. 209–213 (cit. on p. 13).

[WK94] Andreas Weber and Reinhard Klemm. “Economy of Description for Single-Valued Transducers”. In: *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 1994, pp. 607–618 (cit. on pp. 4, 13, 48).