Transductions: resources and characterizations

Olivier Gauwin

Université de Bordeaux
Laboratoire Bordelais de Recherche en Informatique

Transductions: resources and characterizations

Mémoire d'habilitation à diriger des recherches

par

Olivier Gauwin

soutenue le 12 octobre 2020 devant le jury composé de :

Rapporteurs	
Hendrik Jan Hoogeboom	Universitair docent, Leiden University / LIACS
Sebastian Maneth	Heisenberg Professor, Universität Bremen
Jean-Marc Talbot	Professeur, Aix Marseille Université / LIF
Examinateurs	
Bruno Courcelle	Professeur émérite, Université de Bordeaux / LaBRI
Sylvain Lombardy	Professeur, Université de Bordeaux / LaBRI
Sylvain Salvati	Professeur, Université de Lille / CRIStAL

Abstract

Transducers define word-to-word transformations by extending automata with outputs. We study some decision problems related to transducers.

First, we characterize some resources required by any functional transducer implementing a given transformation. We begin with two algorithms determining whether a two-way functional transducer has an equivalent one-way transducer, and synthesizing it in this case. If the transducer is not one-way definable, another algorithm permits to decide whether it can perform its reversals only at the borders of the input word (sweeping transducers), and determine the minimal number of passes over the input. A side result is the minimization of the number of registers of a particular class of streaming string transducers, a model of one-way transducers with registers. We also study the memory required when evaluating visibly pushdown transducers, in particular whether the stack is required, and if so, whether the memory can be bounded by the degree of nesting of the input word.

Second, we study the algebraic properties of functional transductions. A central result is an algorithm that takes a one-way transducer (or a bimachine) as input, and decides whether it belongs to a given decidable congruence class (for instance, aperiodic congruences). A transfer theorem between algebra and logic permits to relate congruence classes with logics. For instance, aperiodic congruences characterize exactly transductions definable in first-order logic. We extend this result to infinite words for the special case of aperiodic transductions. As a consequence, it is decidable whether a rational transduction is first-order definable, for the cases of finite and infinite words.

Foreword

The present document is the manuscript prepared in view of obtaining the "habilitation à diriger des recherches", the French habilitation. It contains a unified presentation of my work related to transducers between 2010 and 2020. This period mostly corresponds to my current position at the University of Bordeaux, at the LaBRI, since 2011. Its content results from three main lines of research, and groups of co-authors:

- the study of *visibly pushdown transducers* exposed in Section 4.2 is a joint work with *Emmanuel Filiot*, *Pierre-Alain Reynier* and *Frédéric Servais*.
- the analysis of *two-way transducers*, presented in Chapter 3 and Section 4.1, was the subject of the PhD thesis of *Félix Baschenis*, that I co-supervised with *Anca Muscholl* and *Gabriele Puppis*.
- the *algebraic characterization of rational transductions*, related in Chapter 5, has been elaborated during the PhD thesis of *Nathan Lhote*, that I co-supervised with *Emmanuel Filiot*, with the active participation of *Anca Muscholl*.

The corresponding publications are listed in Section 1.3. During this period, I also worked on other unrelated topics. I chose not to include them, and opted for a more homogeneous presentation.

Contents

Chapter 1

Introduction

In the last few years, I gradually realized that every research topic finds someone interested in. This sounds naive, but it is somehow reassuring, because for every awkward question that we leave aside, there may be someone finding it not so awkward...

In this introduction, I would like to highlight where the beauty lies around the results exposed in this manuscript, from my point of view.

1.1 Beauty and the Beast

Beauty in computer science. During my studies, three concepts were particularly appealing to me in computer science.

The first one was the lambda calculus, probably because it mixed a small abstract language for writing functional programs, and a simple way to "run" them, on paper. The second one came a few years later, when I learned about Turing machines, and especially the concept of universal Turing machine, explaining for instance virtual machines. The third one is the connection between logic and automata,i.e. how to relate a simple description (through a logical formula) and a way to check it (an automaton), automatically. To me, it was a kind of program synthesis, a very powerful construction.

These three points all relate a "mechanical" aspect of computers and programs, to a "descriptive" aspect, in an automatic way. This is, I think, quite specific to computer science: usually, this is typically a human task to implement a description (specification) into a concrete realization (mechanism).

Beauty in transducers. The results presented in this manuscript concern transducers, that is, finite-state automata enriched with an output word on each transition, and thus associating output words to an input word. They constitute a way to implement a word-to-word transformation, that we name transduction, i.e. the relation defined by the transducer.

One pleasant aspect of transductions comes from their descriptions through *logic*, *algebra*, and *machines* (here, transducers), and equivalences between these models. This is well known for regular languages, that admit equivalent descriptions through monadic second-order logic, finite syntactic congruences or monoids (an algebraic characterization), and finite-state automata [Büc60, Tra61, Myh57, Ner63, RS59], among others. For transducers, this kind of correspondences appeared progressively. For instance, the class of rational functions is captured by one-way functional transducers [Büc60, Sch61, EM65, GR66, Niv68, AHU69, Eil74, Cho77, Ber79], by order-preserving monadic second-order transducers [EH01, CE12, Boj14, Fil15], and by finite left/right syntactic congruences (or bimachines) [RS91]. This will be explained in more details in Chapter 2 and Chapter 5. Beyond these correspondences, rational functions

(and relations) enjoy many other nice properties and representations, as exposed for instance in the textbooks [Ber79, Sak09, CE12].

When I started to work on transducers, I discovered other nice constructions. One of them is described by Hopcroft and Ullman in [HU67], and consists in simulating a deterministic one-way automaton while running a deterministic two-way transducer. In this construction, the two-way transducer is capable of leaving a position to the left, perform leftward moves, and then come back to the initial position. This seems impossible at first sight, and is made possible by the deterministic nature of the simulated one-way automaton. This construction has been recently improved by using reversible transducers [DFJL17]. Other important results on transducers appeared recently, as for instance the decidability of equivalence of deterministic top-down tree-to-string transducers [SMK18], or the class of polyregular functions [Boj18, BKL19].

Transducers: the Beast. Transducers (and, generally, transformations) introduce additional challenges. At the "machine" level, a basic tool in proofs involving automata is to pump automata runs. But transducers have *outputs* on their transitions, and this must be taken into account when pumping. This has several consequences. For instance the equivalence problem for one-way transducers is undecidable [FR68, Gri68, Iba78], and some transducers cannot be determinized.

We will also study *two-way* transducers. Allowing reversals in runs is problematic, even for automata. For instance, it is still open whether two-way non-deterministic automata are exponentially more succinct than deterministic two-way automata [SS78]. On transducers, deciding whether a two-way functional transducer has an equivalent one-way transducer was open until 2013, when it was proved to be decidable [FGRS13].

At the "algebraic" level, languages are characterized by syntactic congruences (for instance). For transformations, the output must also be taken into account. Schützenberger proposed the notion of bimachine [Sch61] for rational transductions (the class defined by one-way functional transducers), and a notion of canonical bimachine, but this one is not minimal, and thus does not convey all algebraic properties of the transduction [RS91]. For two-way functional transducers, no algebraic characterization has been proposed yet.

Beauty and the Beast in our contributions. The results exposed in this manuscript are mainly decision procedures on transducers, that is, algorithms indicating whether a transducer verifies a given property, as for instance being one-way definable, being first-order definable, etc. At this point of the manuscript, it is difficult to exhibit where the beauty lies in these results, but let me try for one result.

A central result in Chapter 3 is an algorithm deciding whether a two-way transducer has an equivalent one-way transducer. As we have seen, analyzing a two-way run is challenging because pumping such a run mixes some parts of the run. We managed to isolate this problem by considering *sweeping* transducers, that only revert their heads at the borders of the input word, as a first goal. This way, we define *inversions* in a run and show that a simple condition on inversions exactly characterizes sweeping transducers having an equivalent one-way transducer. Another nice part of the proof, is the notion of *components* of a loop, that explains how the parts of a run are mixed when a loop is pumped. This is used to adapt the proof to (general) two-way transducers, from the proof on sweeping transducers. These two concepts (inversions and components), and this intermediate stage of sweeping transducers, were circumvented in our first attempt [FGRS13] and now yield a cleaner proof, with a more computationally efficient decision procedure.

1.2 Hands-on

Beyond the beauty of formal language theory, another source of motivation for me is the link between a practical issue and its theoretical aspects.

How much input information is required? Back to my PhD, a first instance of such a link was *earliest query answering*, that is, finding the first position, when reading an XML document, from where the query can be answered. This is required when one wants to use the minimal amount of memory, but it involves a decision procedure, that we managed to infer using an "automaton" approach [GNT09].

In the present manuscript, we will see how to decide whether a two-way functional transducer is definable by a one-way transducer. This looks like a "theoretical" question, but it can also be seen as a way to measure how complex a transformation is: "Do I need to go back in order to perform this transformation with finite memory?". We will see several variants of this question, like "Can I perform this transformation with finite memory by reversing the head only at the borders? And how many reversals do I need in this case? Is there a global bound on the number of reversals required for performing this transformation?". Our algorithms will also synthesize transducers, for instance a one-way transducer from a two-way transducer, whenever it exists.

How much memory is required? In automata theory, a classical question is: how many states are needed by any deterministic automaton recognizing a given language? The answer was given by Myhill and Nerode [Myh57, Ner63] and the minimal automaton obtained is defined by the right syntactic congruence of the language.

When moving to transducers, a first question is whether there exists an evaluation algorithm using bounded memory. This is the case iff there exists a deterministic one-way transducer implementing this transformation, as shown in Chapter 4. Hence, given a functional one-way transducer, one may want to decide whether there exists an equivalent deterministic one. An algorithm was proposed in [Cho77] and improved in [BC02]. Starting from a two-way functional transducer, one can combine the "one-way definability" procedure with that of [Cho77, BC02], in order to decide whether it can be evaluated with bounded memory.

Now, if we have a deterministic one-way transducer, a minimization procedure exists, similar to that of regular languages [Cho03]. In the non-deterministic (functional) case, we will see in Chapter 5 that we need bimachines as deterministic devices, and we can find minimal bimachines (they are not unique anymore, though).

There are other ways to minimize transducers. One of them is the minimization of the number of registers of a *streaming string transducer*, a model of one-way transducer equipped with registers, and as expressive as two-way transducers [AC10]. We do not show it in full generality (which is still an open question), but in the case where registers contents cannot be concatenated (they can only add a constant word to their left and/or their right, not the content of another register).

We also study *visibly pushdown transducers*. These are transducers reading words over a nested alphabet, i.e. letters are either opening or closing. On opening letters, a visibly pushdown transducer can push a symbol on its stack, and pop on closing letters. Opening and closing letters mimic opening and closing tags of XML documents. For these transducers, we study whether the transduction they define can be evaluated with bounded memory, or with a memory bounded by the height (i.e. degree of nesting) of the input word globally, or at any time point.

1.3 Outline

The manuscript is organized as follows (we also list here the related publications).

Chapter 2 introduces the main devices that we will use throughout the manuscript: finite-state transducers, monadic second-order transducers, and streaming string transducers. It enumerates some variants of these devices, the corresponding transduction classes, and the relations between them.

Chapter 3 focuses on the "one-way definability" problem: given a functional two-way transducer, does there exist an equivalent one-way transducer? We first study the existing proofs showing that two-way *automata* can always be translated into one-way automata. Then, we exhibit some properties of two-way *transducers*, especially the combinatorics that will be used in subsequent proofs, and also lower bounds for one-way definability.

Then, we describe two decision procedures for the one-way definability of transducers. The first one follows the Rabin-Scott proof for automata [RS59], is non-elementary, and has been presented at LICS'13 [FGRS13]. The second one is elementary and follows Shepherdson's proof for automata [She59]. For this latter proof, we proceed in two steps. First, we study the case of sweeping transducers, as exposed at FSTTCS'15 [BGMP15]. Second, we lift this proof to arbitrary functional two-way transducers (as presented at LICS'17 [BGMP17]). The whole proof, in two steps, has been published in the LMCS journal in 2018 [BGMP18].

Chapter 4 is devoted to the resource analysis for *evaluating* transductions. The first part tackles the problem of whether a two-way functional transducer has an equivalent sweeping transducer, and if so, determines how many sweeps are needed. It also addresses the minimization of concatenation-free streaming string transducers, by providing back-and-forth translations between this model and sweeping transducers. These results have been presented at ICALP'16 [BGMP16].

The second part focuses on visibly pushdown transducers, in particular how much memory is needed to evaluate them, compared to the height of the input word (its nesting depth). Three classes are exhibited: bounded memory (uniform bound on the memory needed), height-bounded memory (the bound now depends on the height of the input word), and online-bounded memory (at each input position, the memory is bounded by the current height of the input). This result has been presented at FSTTCS'11 [FGRS11]. A full version appeared in LMCS in 2019 [FGRS19].

Chapter 5 is the algebraic part of the manuscript. We analyze the algebraic properties of transductions through some congruences. Our aim is to decide whether a given transduction is inside a class of congruences (for instance, aperiodic congruences). For functions defined by *deterministic* one-way transducers, a minimization procedure [Cho03] exists and preserves algebraic properties. When moving to rational functions (defined by *non-deterministic* functional one-way transducers), we use bimachines to implement the transductions, and especially study the canonical ones, and the minimal ones. We devise an decision procedure for deciding whether a rational function, given as a bimachine, belongs to a (decidable) congruence class. This procedure has been presented at LICS'16 [FGL16b]. For the special case of aperiodic congruences, we exhibit a more direct and more efficient algorithm, as presented at FSTTCS'16 [FGL16a]. We also analyze the links with logics, and establish a transfer theorem. In particular, rational functions with aperiodic congruences are those definable in first-order logic. All these results are gathered in an article published in LMCS, in 2019 [FGL19].

In the second part of the chapter, we describe how these results can be lifted to the case of *infinite* words. The decision procedure we obtain is more specific. It only concerns the class of aperiodic congruences, not any decidable congruence class. The case of infinite words requires additional developments, for instance we introduce an intermediate class of transductions (quasi-sequential transductions), and also two new congruences. This work has been presented at FSTTCS'18 [FGLM18].

Chapter 6 concludes this manuscript. We briefly list the results presented in this manuscript, and propose some perspectives related to them in more details.

Chapter 2

Transducer models

Contents

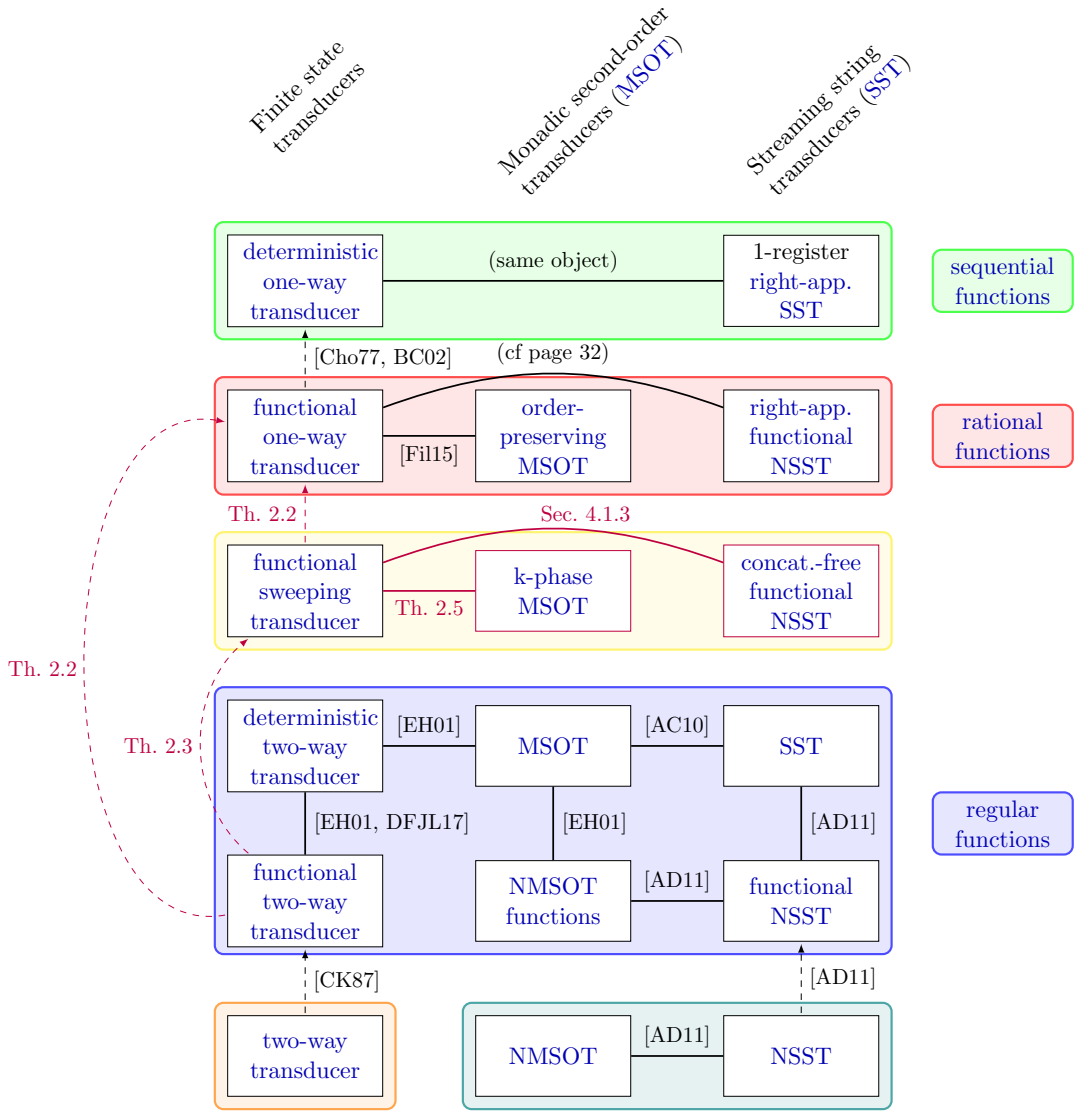
This chapter introduces the main devices used in this manuscript for defining transformations from words to words. We also refer the reader to the recent surveys [FR16, MP19b] on this topic. We postpone the *algebraic* view to Chapter 5, and focus here on the *automata* and *logic* views.

Regular languages machinery. The devices that we will use for transformations are all derived from language acceptors (logics or automata), and inherit some of their properties.

Two-way automata have been introduced by Rabin and Scott in the seminal paper [RS59]. In Section 3.1, we list various proofs showing that they are not more expressive than one-way automata, as first proved in this seminal paper, and also by Shepherdson in [She59]. *Monadic second-order logic (*MSO*)* also originates from the 1950s, and the first direct link between MSO and one-way automata has been quickly identified [Büc60, Tra61]. These results are among the numerous characterizations of this robust class of languages, the *regular languages*.

Regular languages enjoy many characterizations and properties, and they probably constitute the most studied class of formal languages. However, from the "machine" point of view, some questions remain open. This is particularly true concerning two-way automata[1]. For instance, it is still open whether non-deterministic two-way automata are exponentially more succinct than deterministic two-way automata [SS78][2].

[1] This is part of the *minicomplexity* framework.
[2] This is the case for sweeping automata [Sip80], for instance.



Colored boxes indicate classes of transformations. Solid lines relate models inside classes, while dashed arrows indicate definability procedures between classes. Contributions appear in purple.

Figure 2.1: Models of word-to-word transformations.

Three transformation models. In this manuscript, we focus on three models of word-to-word transducers. These models (and their variants) are depicted in Figure 2.1, by column:

- *finite-state transducers* are the "transducer" extension of automata. One-way transducers started to be studied during the 60s [EM65, AHU69, AU70]. One of the first papers on *two-way* transducers is [EY71], but Shepherdson already noticed that two-way transducers are strictly more expressive than one-way transducers in [She59, Note 4].
- *MSO transducers* (MSOTs) have been introduced by Courcelle [Cou94, CE12]. They rely on MSO formulas to relate an input structure (here, a word) to an output structure.
- *streaming string transducers (SSTs)* are one-way transducers equipped with registers [AC10]. These registers contain words (over the output alphabet) that can be combined

and extended when firing transitions, to form the final output word.

Classes of transformations. Each of these models comes with a number of variants, according to the features we allow in the model: functionality, determinism, reversals, etc. This chapter lists the corresponding classes, the relations between them, and the related definability problems. These are also illustrated in Figure 2.1. Let us give a first overview on the main classes, starting from the less expressive:

- *sequential functions* are defined by one-way deterministic finite-state transducers (also called generalized sequential machines, GSMs). These machines can also be viewed as SSTs with only one register, and always concatenating to the right in this register (this property is named *right-appending*). There is no known restriction on MSOT capturing this class.

- *rational functions* are defined by functional one-way finite-state transducers, hence allowing non-determinism. They are also captured by functional right-appending non-deterministic SSTs, and by MSOTs with the *order-preserving* restriction, stating that every move in the output structure is rightwards.

- *sweeping functions* define an intermediate class between one-way transducers (rational functions) and two-way transducers (regular functions, hereafter), as they are recognized by two-way transducers allowing reversals only at the borders of the input word. This class plays an important role in some of our proofs: our "one-way definability test" is first explained on sweeping transducers, before being adapted to any two-way transducer. Moreover, we provide characterizations in term of MSOT restriction, and in terms of SSTs. It also coincides with the class of functions that can be implemented by a two-way transducer with a uniform bound on the number of reversals of its runs. A characterization by regular function expressions has also been exhibited in [BR18].

- *regular functions* are characterized by deterministic two-way transducers, MSOTs and SSTs, hence their name. They are also captured by their "functional non-deterministic" versions: functional two-way transducers, functional NMSOTs, and functional NSSTs (defined hereafter).

- beyond functions, these devices can be used to recognize relations. The extensions of two-way transducers and SSTs to the non-deterministic case are natural. For MSOTs, this corresponds to adding second-order parameters, that are fixed before being interpreted on the input structure. This time, the expressiveness does not coincide: *two-way transducers* form one class of relations, and NMSOTs and NSSTs a second class, and these two classes are incomparable.

Other devices used in this manuscript. In Chapter 5, we will introduce another device capturing rational functions, namely bimachines [Sch61]. These are similar to one-way deterministic transducers with a (co-deterministic) look-ahead. Moreover, variants of transducers using a stack will be introduced and studied in Section 4.2.

Other means to define transformations. There are numerous ways to define transformations, and enumerating them would require a whole thesis. Let us mention some of them, related to the questions addressed in this manuscript.

Rewriting systems provide an alternative way of defining a transformation. They usually start from a word, and apply rewriting rules to obtain new words. Close links between two-way

transducers, *tree transducers*, some rewriting systems and some *grammars* have been established [Raj72, ERS80], and also with pushdown automata [EY71]. *Regular function expressions* [AFR14, DGK18, BR18] provide a "regular expression" mechanism for defining word-to-word functions, and capture exactly regular functions.

Of course many other questions arise when trying to extend the kind of *structures* under consideration. For instance, streaming string transducers have been extended to operate on *infinite strings* [AFT12], with similar logics-automata connections, on *trees* [AD17] and *quantitative languages* [ADD$^+$13, AFM$^+$20].

Transducers are not the only possible extension of automata for defining relations on words. One can also recognize pairs of words using an *automaton with two tapes*: one for the input word and the other for the output word [Ber79, HK91, PS99]. Carton [Car12] studied the model where the heads of the two tapes are two-way, but move synchronously. The first tape is read-only and the second one write-only, with the possibility to write a word (not only a letter) in the current cell of the output tape. It is proved that, in the non-deterministic case, this model coincides with rational relations, while, in the deterministic case, it coincides to rational functions (this is more surprising). In other terms, there always exists an equivalent one-way transducer. Another model consists in having two two-way read-only tapes that are not synchronous anymore. When firing a transition, a letter is read on each tape, and the chosen transition indicates the head direction for each head. This model is introduced and studied in [CES17], together with its alternating extension.

In [RV19], Reynier and Villevalois extend one-way transducers by allowing, on each transition, two output words. One is prepended to the current output, and the other one appended. This defines an intermediate class, between rational and regular functions (and relations).

Other questions related to transformations. With each model of machine comes the question of minimization, i.e. finding an equivalent machine of minimal size. We already mentioned that these questions are still open for two-way machines (minicomplexity). We will mention this question for rational functions in Chapter 5.

Each class also defines some specific decision problems, typically deciding the *equivalence* of the transformations defined by two devices of that class. In this chapter, we will cite the related results when defining the classes, even though this problem is usually independent from the definability of one class in another.

Another point of view is that of the *output language*: Each device can be considered as a machine only producing output words, and thus each class of device comes with a class of output languages. For instance a deterministic two-way transducer can produce the language $\{a^n b^n c^n \mid n \geq 0\}$, which is not context-free, but some context-free languages cannot be produced by any deterministic two-way transducer [Eng81][3]. The output languages of deterministic two-way transducers are known to be exactly those of matrix grammars of finite index [Raj72], and also to that of EDT0L of finite index [Lat77]. Some iteration lemmas for this class of languages have been proved [Roz85, Roz87, Smi14], but are not sufficient to decide one-way definability, for instance (see Chapter 3).

Alternative semantics. In 2014, Bojańczyk proposed a new semantics for transducers [Boj14]. Instead of interpreting a transducer as a machine recognizing pairs of input/output words, he proposed to keep track of the link between each output position and the input position where it has been generated. This is called the *origin* semantics of transducers, and is well-defined

[3] Amusingly, the Dyck language with only one type of parentheses is the output of a non-deterministic two-way transducer, see for instance [Roz85].

on finite-state transducers, as well as MSO transducers. Many decision problems become easier with this semantics [BMPP18, FMRT18], and new characterizations appeared [BDGP17, DFL18].

The origin semantics is quite restrictive, as it imposes that two transducers are equivalent iff each output letter is produced at the very same input position. A way to relax this strong property is to use *resynchronizers*, as introduced in [FJLW16] (see also [FL15]). Instead of considering a transduction as a set of pairs of input/output words, they can be defined as a set of words mixing input and output letters, where each letter is typed (input or output). A resynchronizer is a transducer operating on such words. Hence it allows to change the origin information in a controlled way. First results on the decision problems related to resynchronizers appeared in [FJLW16, BMPP18, DFP18, DFF19], and also in [BKM+19], with additional results on the synthesis of resynchronizers.

2.1 Finite state transducers

2.1.1 Words, languages and relations

Words and languages An *alphabet* is a finite set, which elements are called *letters*. Unless otherwise stated, we always assume that an alphabet contains at least two elements. A *word* over an alphabet Σ is a finite sequence $u = a_1 a_2 \cdots a_n$ of letters $a_i \in \Sigma$, and we denote by Σ^* the set of all words over Σ, including the empty word ϵ, and Σ^+ when excluding the empty word. The length n of the word u is written $|u|$. A word $u = a_1 a_2 \cdots a_n$ has *period* p if for every i such that $1 \leq i \leq |u| - p$, we have $a_i = a_{i+p}$. A word $v \in \Sigma^*$ is a *prefix* of u, denoted by $v \preceq u$, if $u = vw$ for some word $w \in \Sigma^*$. In this case we denote by $v^{-1}u$ this word w. We write $u \wedge v$ for the *longest common prefix* of u and v. The longest common prefix of a set L of words is denoted $\bigwedge L$. We define the *delay* $\mathsf{del}(u, v)$ between two words u, v as the pair (u', v'), such that $(u, v) = (wu', wv')$ where $w = u \wedge v$.

We associate with u its *domain* $\mathrm{dom}(u) = \{1, \ldots, n\}$, and will mostly use it in order to define the logical structure associated with a word. In that setting, we name *domain position* an element of $\mathrm{dom}(u)$. We will need another notion to describe the locations *between* letters where states of the automata will be assigned. We name *positions* such locations, and write $\mathrm{pos}(u)$ for the set $\{0, 1, \ldots, n\}$ of positions of u: 0 is placed before the first letter of u, n is after the last one, and i is between a_i and a_{i+1}, for every $1 \leq i < n$. These notions are illustrated in Figure 2.2. This way we naturally define the *factor* $u[i, j]$ of u between two positions i and j. We write $u[i]$ for the ith letter of u. The *mirror* of a word is obtained by reading it from right to left, that is: $\mathrm{mirror}(a_1 a_2 \cdots a_n) := a_n \cdots a_2 a_1$ with $a_i \in \Sigma$.

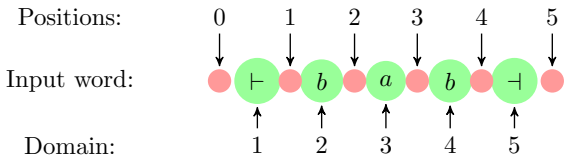


Figure 2.2: Positions and domain of the word $\vdash bab \dashv$.

On two-way devices (automata and transducers), input words will always begin with the special letter $\vdash$ and end with the special letter $\dashv$. These letters are used by the device to identify the borders of the word. They are part of the alphabet, but only appear at the borders. Hence any word (on two-way devices) is of the form:

$$a_1 a_2 \cdots a_n \qquad \text{with } a_1 = \vdash, a_n = \dashv, \text{ and } a_i \notin \{\vdash, \dashv\} \text{ for all } 1 < i < n$$

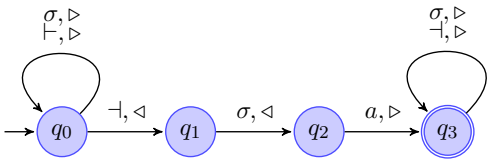


Figure 2.3: A two-way automaton $\mathcal{A}$.

A *language* over an alphabet Σ is a set of words over Σ. In Chapter 5 we will also study languages of infinite words. We defer their definition to that chapter.

Relations and functions In this manuscript we mainly focus on transformations from words to words. In their most general form, we will study *relations* where a single word $u \in \Sigma^*$ can be mapped to any number of words $v \in \Delta^*$. Hence such a relation $\mathcal{R}$ is a set of pairs: $\mathcal{R} \subseteq \Sigma^* \times \Delta^*$. The *domain* of $\mathcal{R}$ is the set of words $u \in \Sigma^*$ for which there exists $v \in \Delta^*$ verifying $(u, v) \in \mathcal{R}$, and is denoted $\mathrm{dom}(\mathcal{R})$. A relation $\mathcal{R}$ is *functional* if for every $u \in \mathrm{dom}(\mathcal{R})$, there exists only one v such that $(u, v) \in \mathrm{dom}(\mathcal{R})$. In this case we consider $\mathcal{R}$ as a function, and write $\mathcal{R}(u)$ for the word v such that $(u, v) \in \mathcal{R}$. Hence the functions we consider in this manuscript are *partial* functions of type $\Sigma^* \to \Delta^*$. We also name them *transductions*, independently of the way they are defined (through a transducer, formula, bimachine, etc). We will generalize them to infinite words in Section 5.2.

2.1.2 Finite state automata

Two-way automata A *two-way automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \vdash, \dashv, \delta, I, F)$, where:

- Q is a finite set of *states*,
- Σ is an alphabet, including special letters $\vdash$ and $\dashv$,
- $\delta \subseteq Q \times \Sigma \times Q \times \{\triangleleft, \triangleright\}$ is the *transition* relation,
- $I \subseteq Q$ is the set of *initial states*, and
- $F \subseteq Q$ is the set of *final states*.

The *size* of a two-way automaton $\mathcal{A}$ is $|\mathcal{A}| = |Q| + |\delta|$. Runs of two-way automata enjoy an intuitive two-dimensional representation, that can be used in order to define them.

Example 2.1. *Consider the two-way automaton $\mathcal{A}$ depicted in Figure 2.3, where σ denotes any letter that does not contradicts determinism. This automaton checks whether the last-but-one letter of a word is an "a". This is a typical example where deterministic one-way automata need an exponential number of states (when considering the family of languages increasing the distance between the letter to check and the end), while deterministic two-way automata remain of linear size w.r.t. this distance. A run of $\mathcal{A}$ on the input word $\vdash aab \dashv$ is illustrated in Figure 2.4.*

A *run* ρ of a two-way automaton $\mathcal{A}$ is a series of points (x, y) that we name locations, labelled with states, and connected by transitions. Formally, a *location* is a pair $\ell = (x, y)$ where x is a position of the input word u (ranging from 0 to $|u|$) and y is a non-negative integer that we call *level*, that denotes the number of times the position x has been previously reached in ρ. As a consequence, rightward transitions lead to a location at an *even* level, while leftward transitions lead to a location at an *odd* level.

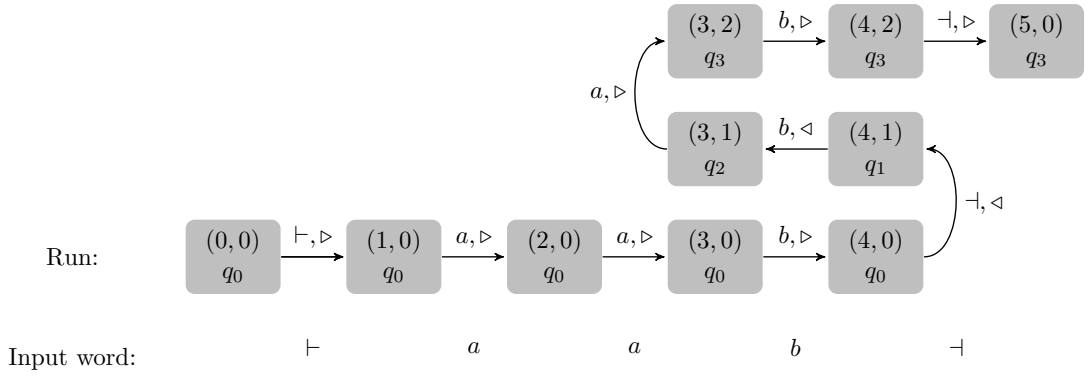
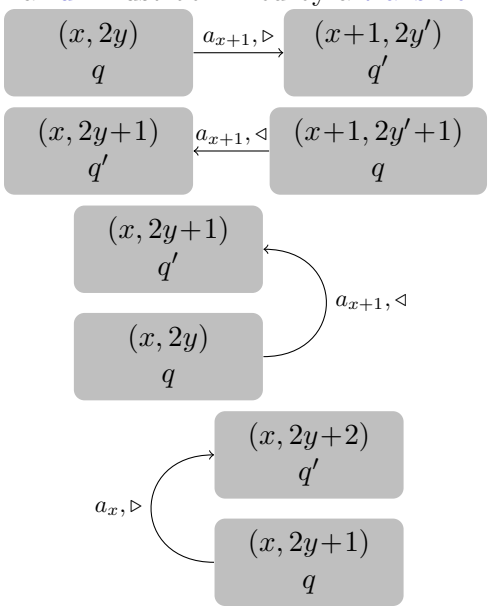


Figure 2.4: Run of $\mathcal{A}$ on the word $\vdash aab \dashv$.

Each location ℓ of a run ρ is associated with a state, that we write $\rho(\ell)$. Consecutive locations in a run must be linked by a transition of the automaton in the following manner:



if $(q, a_{x+1}, q', \triangleright) \in \delta$

if $(q, a_{x+1}, q', \triangleleft) \in \delta$

if $(q, a_{x+1}, q', \triangleleft) \in \delta$

if $(q, a_x, q', \triangleright) \in \delta$

This discards leftward transitions on $\vdash$, and also additional transitions once a rightward transition on $\dashv$ has been performed.

A run ρ is *successful* if it starts at location $(0, 0)$, ends at location $(|u|, 0)$, and $\rho(|u|, 0)$ is a final state. We also define the total order $\trianglelefteq$ on the locations of a run ρ by letting $\ell \trianglelefteq \ell'$ iff ℓ appears before ℓ' in ρ. The *language of* $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words for which a successful run of $\mathcal{A}$ exists.

We name *reversal* of a run any transition increasing the level by 1. A two-way automaton is *sweeping* if it performs reversals only at the borders ($\vdash$ and $\dashv$) of input words. It is *k-sweeping* (for $k \in \mathbb{N}$) if it is sweeping and performs at most k reversals per successful run. A two-way automaton is *deterministic* if it has a single initial state, and it does not contain two transitions (q, a, p, d) and (q, a, p', d') with $(p, d) \neq (p', d')$. It is *unambiguous* if if admits at most one successful run per input word.

One-way automata A *one-way automaton* (or *automaton* for short) is a special kind of two-way automaton, containing only rightward transitions. For convenience, we usually omit the direction in the transitions, thus in $Q \times \Sigma \times Q$. Moreover, end-markers $\vdash$ and $\dashv$ are only added when compared with two-way automata. We also sometimes consider a run of a one-way

automaton as a series of states, instead of a series of locations associated with a state.

Language classes In Section 3.1 we review some constructions showing that any two-way automaton is equivalent to some one-way automaton. This means that these two classes have the same expressivity.

Moreover, any one-way automaton can be determinized using the well-known subset construction. So, neither non-determinism nor two-wayness increases the expressive power of automata, and they all coincide with the class of *regular languages*. We will see that the setting is different when moving to transducers.

2.1.3 Finite state transducers, and relation classes

While automata recognize languages of words, transducers recognize *relations* between words. If we denote the input alphabet by Σ, and the output alphabet by Δ, the transducer will define a relation in $\Sigma^* \times \Delta^*$.

Formally, a *two-way transducer* $\mathcal{T}$ is a tuple $(\mathcal{A}, \mathsf{out}, \iota)$ where $\mathcal{A} = (Q, \Sigma, \vdash, \dashv, \delta, I, F)$ is a two-way automaton, $\mathsf{out} : \delta \to \Delta^*$ maps every transition of $\mathcal{A}$ to an output word, and $\iota : F \to \Delta^*$ associates an output word with every final state. We name $\mathcal{A}$ the *underlying automaton* of $\mathcal{T}$, out the *output function* of $\mathcal{T}$, and ι the final output function of $\mathcal{T}$. For convenience we will identify the *states*, *transitions*, *configurations*, and *runs* of a transducer with those if its underlying automaton.

Given a run $\rho = \ell_0, \ell_1, \ell_2, \dots, \ell_n$ of $\mathcal{T}$ (and thus $\mathcal{A}$) on u, we write $\mathsf{out}(\rho)$ for the output of ρ on u, defined by $\mathsf{out}(t_1)\mathsf{out}(t_2)\cdots\mathsf{out}(t_n) \in \Delta^*$, where t_i is the transition used between ℓ_{i-1} and ℓ_i in ρ. The *relation* associated with $\mathcal{T}$, denoted $[\![\mathcal{T}]\!]$, is the set of pairs (u, v) for which there exists a successful run ρ of $\mathcal{A}$ on u, and $v = \mathsf{out}(\rho)\iota(q)$ where q is the target state of the last transition of ρ.[4] Two transducers are *equivalent* if they define the same relation. The *domain* of $\mathcal{T}$, denoted $\mathrm{dom}(\mathcal{T})$, is $\mathcal{L}(\mathcal{A})$. The *size* of a transducer is the size of its underlying automaton plus the size of its output function (it contains in particular the size of the output words of this function).

A two-way transducer $\mathcal{T}$ is:

- *one-way* (resp. *deterministic*, *unambiguous*, *sweeping*, *k-sweeping*) if its underlying automaton is one-way (resp. deterministic, unambiguous, sweeping, k-sweeping),
- *functional* if the relation it defines is functional.

Deterministic one-way transducers were also called *input-deterministic* transducers, or, when ι maps every final state to ϵ, *generalized sequential machines (GSMs)* [Eil74, Gin66]. Functional transducers are sometimes called *single-valued* in the literature. We only consider *real-time* transducers, i.e. transducers that do not have ϵ-transitions [Sak09]. Two-way transducers are closed under composition [CJ77].

Relation classes In the following, we say that a relation $R \subseteq \Sigma^* \times \Delta^*$ is *implemented* by a transducer $\mathcal{T}$ if $[\![\mathcal{T}]\!] = R$. Several classical transformation classes are based on the transducers defined in this section, as depicted in Figure 2.1:

- *regular relations* (resp. functions) are the relations (resp. functions) for which there exists a two-way transducer implementing it.

[4] We do not include $\iota(q)$ in the definition of $\mathsf{out}(\rho)$ in order to be able to concatenate runs.

- *sweeping relations* (resp. functions) are relations (resp. functions) having a sweeping transducer implementing it (idem for k-sweeping). We name those relations sweeping definable (or k-sweeping definable, when k is given).

- *rational relations* (resp. functions) are those for which there exists a one-way transducer implementing it. For this reason we also name such relations one-way definable.

- *sequential functions* are functions that can be implemented by a deterministic one-way transducer (any relation defined by a deterministic transducer is functional, by definition).

Hence, we call sequential functions what is named "subsequential functions" in [Sch77], as it associates output words with final states. Indeed this class captures the most "well-behaved" notion of sequentiality between both [LS06]. This clearly appears at two places in this manuscript: sequential functions are exactly those that can be evaluated with bounded memory (Proposition 4.2), and have a direct characterization in terms of congruence (Section 5.1.2).

In this manuscript we mainly focus on *functional* transformations. The main reason is that the decision problems we address are often decidable for functions, but undecidable for relations. We also preferably stick to the "machine-oriented" terminology rather than the class mentioned above. For instance we preferably write one-way definable, rather than rational.

In Chapter 5, we use bimachines to define rational transductions, instead of using one-way transducers. We defer their definition and analysis to that chapter, as they do not define another class of transformations.

2.1.4 Definability problems

When two classes of relations $\mathcal{C}_1$ and $\mathcal{C}_2$ are introduced, they naturally come with the following *definability problem*:

> Given a relation in $\mathcal{C}_1$, is there an equivalent relation in $\mathcal{C}_2$?

For this decision problem to be well defined, each class $\mathcal{C}$ must come with an explicit description of its relations, and this will always be the case for the classes defined in this manuscript (through transducer models, formulas, finite congruences, etc). Let us review the definability problems related to these classes.

Relations When considering relations, many decision problems are undecidable, notably universality (decide whether a given transducer implements $\Sigma^* \times \Delta^*$) and thus equivalence (decide whether two given transducers define equivalent relations) [FR68, Gri68, Iba78]. However, functionality of two-way transducers is decidable [CK86] in PSpace. Indeed, this problem can be reduced to the emptiness of a one-counter automaton, that tries to build (non-deterministically) a witness of non-functionality. Such a witness is composed by an input word, and two runs of the transducer, such that either the outputs of the runs differ in length, or they differ at a particular position. In both cases this can be checked with a one-counter automaton.[5]

Concerning definability by a one-way transducer, we proved that this is also undecidable, even when restricting to sweeping relations:

Theorem 2.1 ([BGMP15, BGMP18])**.** *It is undecidable whether a given sweeping transducer is one-way definable.*

[5] A similar approach will be used for the proof of Theorem 4.5.

The proof of this theorem is based on an encoding of the Post Correspondence Problem (PCP), and was inspired by the proof by Ibarra that the equivalence of one-way transducers is undecidable [Iba78].

Consider the instance of PCP given by the alphabets Σ and Δ and the two morphisms $f, g : \Sigma^* \to \Delta^*$. Recall that this instance is a PCP solution iff there exists $w \in \Sigma^+$ such that $f(w) = g(w)$. We call *encoding* every pair $(w \cdot u, w \cdot \#^n)$ with $w \in \Sigma^*$, $u \in \Delta^*$, and $n \in \mathbb{N}$. We call *good encodings* those verifying $n = |u|$ and $u = f(w) = g(w)$, and *bad encodings* the others (but still encodings). Bad encodings can be recognized by a sweeping transducer, by guessing and checking the various ways in which an encoding can fail.

The reduction is based on the fact that this sweeping transducer is one-way definable iff the corresponding PCP instance has no solution. On one hand, its is quite easy to see that if there is no solution to the PCP instance, all encodings are bad, and the relation is one-way definable. The most technical part is to prove that if there is a solution to the PCP instance, the sweeping transducer is not one-way definable. This is obtained by noting that if w is a solution, then w^n also is. This permits to use pumping arguments towards a contradiction, when the transducer is supposed one-way definable.

Functional transducers: two-way to one-way When restricting to functions instead of relations, we retrieve decidability:

Theorem 2.2 ([BGMP15, BGMP17, BGMP18])**.** *It can be decided in* 2ExpSpace *whether a given functional two-way transducer is one-way definable (in* ExpSpace *if the two-way transducer is known to be sweeping).*

If so, an equivalent one-way transducer can be built in 3ExpTime *(in* 2ExpTime *for sweeping transducers).*

This constitutes the central result presented in Chapter 3, therefore we defer the details to this chapter. We will refer to the decision problem as *one-way definability*. Typical examples of functions that are two-way definable but not one-way definable are provided in the following example.

Example 2.2. *The following functions can be implemented by a two-way transducer, but not by a one-way transducer (assuming $|\Sigma| > 1$):*

- $f_{\text{copy}} : \Sigma^* \to \Sigma^*$ *such that* $f_{\text{copy}}(u) = uu$,
- *the* mirror *function over* Σ^*.

Functional sweeping transducers Sweeping transducers define an intermediate level between one-way and two-way transducers[6], and the related decision problems are also decidable in the functional case.

Theorem 2.3 ([Bas17, BGMP16])**.** *It can be decided in* 2ExpSpace *whether a functional two-way transducer has an equivalent sweeping transducer.*

Moreover, for a given $k \in \mathbb{N}$ and a given functional two-way transducer, it is decidable in 2ExpSpace *whether there exists an equivalent k-sweeping transducer. If the input transducer is already sweeping, then the problem is decidable in* ExpSpace.

[6] even with unary alphabets [Gui16a]

These results will also be presented in more details in Section 4.1. As we will see, each of these results also comes with an algorithm building an equivalent transducer, when it exists (in 3ExpTime for 2ExpSpace decision problems, and 2ExpTime for ExpSpace ones).

Another interesting link between models (presented in Section 4.1) is the following: Functional transductions that can be implemented by a sweeping transducer are exactly those that can be implemented by a *bounded-reversal* two-way transducer, that is, a two-way transducer with a universal bound on the number of reversals per run.

Determinism Let us now consider definability problems related to determinism. For functional two-way transducers, the question is irrelevant, as every functional two-way transducer can be determinized (i.e. has an equivalent deterministic two-way transducer). A first sketch of proof appeared in [Eng81], and a full proof in [EH01].[7]

The uniformization framework generalizes this result to *non-functional* two-way transducers: For every two-way transducer $\mathcal{T}$, there exists a deterministic two-way transducer $\mathcal{T}'$ with the same domain, such that $[\![\mathcal{T}']\!] \subseteq [\![\mathcal{T}]\!]$. The proof proposed by De Souza [dS13], provides an upper bounded of four-exponential time. This proof is based on the nice construction by Hopcroft and Ullman [HU67], that allows a deterministic two-way transducer to simulate a deterministic one-way automaton: While it is straightforward rightwards, any leftward move of the two-way transducer may induce non-determinism in the one-way automaton. This non-determinism can be lifted by entering a leftward-then-rightward mode that modifies the run of the initial two-way transducer, and is able to find the exact position where it entered this leftward-then-rightward mode (which seems impossible at first sight).

In the functional case, this result has been strengthened recently, by showing that one can always find a deterministic *and co-deterministic* (that is, deterministic when transitions are interpreted in reverse) two-way transducer (called *reversible*) as uniformizer, from any functional two-way transducer [DFJL17]. The main tool of this construction is a clever way to build a reversible transducer from a co-deterministic one. It is obtained by considering the tree of runs of the codeterministic transducer on an input, and exploring it using two states of the two-way transducer. Using this technique, the construction of the reversible transducer from a functional two-way transducer only requires exponential time, while it required four-exponential time in [dS13].

When considering one-way transducers, determinism adds a lot of constraints, as only a bounded amount of output can be stored in the memory (states), when a delay is necessary to produce it.

Example 2.3. *The following function can be defined by a one-way transducer, but not by a deterministic one:*

- $f_{\text{last}} : \Sigma^+ \to \Sigma^+$ *such that* $f_{\text{last}}(ua) = au$, *for all* $a \in \Sigma$ *and* $u \in \Sigma^*$

Theorem 2.4 ([Cho77, BC02, MP19b])**.** *Deciding whether a given functional one-way transducer has an equivalent deterministic one-way transducer is* NLogSpace*-complete.*

While prior characterizations of sequential functions were known [Gin66, Sch77], the decidability was first proved by Choffrut [Cho77] by showing that deciding sequentiality amounts to analyze some regular properties of the outputs produced by input words of bounded length. This can be checked syntactically: A functional one-way transducer $\mathcal{T}$ satisfies the *twinning property* if for all $u_1, u_2 \in \Sigma^*$, for all $v_1, v_2, w_1, w_2 \in \Delta^*$, for all initial states q_0, q_0', and for all co-accessible states $q, q' \in Q$,

[7] Checking equivalence of deterministic two-way transducers is decidable in PSpace [Gur82, CK86].

$$\text{if } \begin{cases} q_0 \xrightarrow{u_1/v_1}_{\mathcal{T}} q \xrightarrow{u_2/v_2}_{\mathcal{T}} q \\ q'_0 \xrightarrow{u_1/w_1}_{\mathcal{T}} q' \xrightarrow{u_2/w_2}_{\mathcal{T}} q' \end{cases} \quad \text{then } \mathsf{del}(v_1, w_1) = \mathsf{del}(v_1 v_2, w_1 w_2).$$

This pattern can be checked in PTime [WK95, BC02]. Its formulation is simple enough to be formulated in a pattern logic ensuring decidability in PTime [FMR18]. Muscholl and Puppis showed that the problem is in fact NLogSpace-complete [MP19b].

When we start from a two-way transducer, we can combine the two results:

1. first decide whether it is equivalent to a one-way transducer and if so, build it (in 3ExpTime, Theorem 2.2),

2. decide whether this one-way transducer can be determinized (in PTime, Theorem 2.4).

As a consequence:

Corollary 2.1. *It is decidable in* 3ExpTime *whether a functional two-way transducer has an equivalent deterministic one-way transducer.*

An open question arises concerning the complexity of this problem. The 3ExpTime upper bound is far from the ExpTime lower bound that comes from the usual lower bound on automata determinization:

Open problem 1 (*Functional two-way to deterministic one-way transducer*)

Determine the precise complexity of the following problem: Given a functional two-way transducer, is it equivalent to some deterministic one-way transducer?

We will see in Section 4.2.1 that functions defined by deterministic one-way transducers correspond exactly to those that can be evaluated with bounded memory, so this question is of particular interest.

2.2 Logics for transformations

In this manuscript, we use logical formulas to express:

- properties over *words*, for instance "a word has even length",
- *word-to-word transformations*, for instance "every position labelled by a is now labelled by b".

In full generality, the logics we present are interpreted over logical structures, that is, a finite domain, and a fixed set of relations interpreted over this domain. In the case of words, the domain is the set of domain positions of the word, and the relations are typically one unary relation ϕ_a for each $a \in \Sigma$, and a binary relation $\leq$ reflecting the (total) order on the domain of the word. We simplify the definitions by instantiating them on words only.

2.2.1 Logics defining word languages

Our basic logic over words will be the *monadic second-order logic* (MSO).

MSO syntax *Formulas* ϕ of MSO are those yielded by the following grammar:

$$\phi ::= \phi \wedge \phi' \mid \neg\phi \mid \exists x.\ \phi \mid \exists X.\ \phi \mid x \in X \mid \mathrm{lab}_a(x) \mid x \leq y$$

where $x, y, \ldots$ denote the first-order variables, $X, Y, \ldots$ denote the second-order variables, ϕ, ϕ' are MSO formulas, and a is a letter from a fixed alphabet Σ.

From this core syntax, we add the usual syntactic sugar, namely parentheses, universal quantifiers and other Boolean connectives:

$$\begin{array}{ll} \forall x.\ \phi \text{ for } \neg\exists x.\ \neg\phi & \phi \vee \phi' \text{ for } \neg(\neg\phi \wedge \neg\phi') \\ \forall X.\ \phi \text{ for } \neg\exists X.\ \neg\phi & \phi \to \phi' \text{ for } \neg\phi \vee \phi' \end{array}$$

and also:

$$\top \text{ for } \forall x.\ (\mathrm{lab}_a(x) \vee \neg\mathrm{lab}_a(x)) \qquad \text{and} \qquad \bot \text{ for } \neg\top$$

MSO semantics We informally present the semantics of MSO formulas. For a formal presentation, we refer the reader to [Str94], which contains a nice yet precise definition in the case of words. We use the standard notion of *free* and *bound* variables of a formula (here for first-order and second-order variables), and the notion of *closed* formula.

An MSO formula ϕ is interpreted over a word $w \in \Sigma^*$: if the word w *satisfies* the formula ϕ we write $w \models \phi$. Let us now explain what this means. First-order variables $x, y, \ldots$ denote domain positions of the word, while second-order variables $X, Y, \ldots$ denote *sets* of domain positions of the word. Hence an interpretation comes with an assignment of variables (that can also be viewed as an annotation of domain positions with variables [Str94]), and:

- $\mathrm{lab}_a(x)$ holds true if x is assigned to a domain position of w labelled by a,
- $x \leq y$ holds true if x is assigned to a domain position at the left of the domain position assigned to y (or to the same domain position),
- $x \in X$ holds true if x is assigned to a domain position that belongs to the set of domain positions assigned to X,
- quantifiers and Boolean connectives are interpreted as usual.

For instance the formula $\exists x.\exists y.\ \mathrm{lab}_a(x) \wedge \mathrm{lab}_b(y) \wedge x \leq y$ holds true on words having an a at the left of a b (possibly with other letters in between).

This way, any closed MSO formula ϕ defines the language of words on which it holds true. We denote this language by $[\![\phi]\!]$, and say that ϕ *recognizes* $[\![\phi]\!]$. We usually write $\phi(x_1, \ldots, x_n, X_1, \ldots, X_N)$ to denote that $x_1, \ldots, x_n$ and $X_1, \ldots, X_N$ are the free variables in ϕ. We will also write $w \models \phi(p_1, \ldots, p_n, P_1, \ldots, P_N)$ whenever ϕ holds true on w when each variable x_i (resp. X_i) is assigned to the domain position p_i (resp. to the set of domain positions P_i).

Language classes As discussed in the introduction of this chapter, the set of languages recognized by closed MSO formulas is exactly the set of languages recognized by one-way automata [Büc60, Tra61], and is called the set of *regular languages*.

The most standard subclass of MSO is the *first-order logic* (FO), i.e. the set of formulas involving no second-order variables. This will be used in Chapter 5 to define a corresponding subclass of one-way transducers.

2.2.2 Logics defining word-to-word transformations

While the standard presentation of MSO originates from the 60's, it has been adapted in the 90's by Courcelle in order to define *transformations* from logical structures to logical structures using MSO transducers [Cou94, CE12]. Let us present this framework on the particular case of word-to-word functions.

MSO transducers The key idea is to use MSO formulas with free first-order variables that will be interpreted over a fixed number of copies of the input word, in order to introduce new relations that will define the output word.

Hence an *MSO transducer* (MSOT) is a tuple composed by:

- a number k of copies of the input word,
- an MSO formula ϕ_{dom} defining the domain of the function,
- k MSO formulas $\phi^i_{\text{pos}}(x)$, with $1 \leq i \leq k$. These formulas describe the domain of the output word, among the k copies of the input w: the domain position p of the ith copy will be in the output if $w \models \phi^i_{\text{pos}}(p)$.
- k MSO formulas $\phi^i_{\text{lab}_a}(x)$, with $1 \leq i \leq k$. They characterize the *labels* of the output word: the domain position p of the ith copy will be labelled by a if $w \models \phi^i_{\text{lab}_a}(p)$.
- k^2 MSO formulas $\phi^{i,j}_{\leq}(x, y)$, with $1 \leq i, j \leq k$. These formulas define the *order* on the domain of the output word. Hence the domain position p of the ith copy will be to the left of the domain position p' of the jth copy if $w \models \phi^{i,j}_{\leq}(p, p')$.

Such a definition allows the output structure to be more general than a word, for instance having several labels at a given domain position, or having an order relation $\leq$ that is not a total order. However, given an MSO transducer, it is decidable if it defines a word-to-word function [Fil15]. In this document we consider only MSO transducers that define word-to-word transformations.[8]

MSO transducers can be extended to *non-deterministic MSO transducers* (NMSOTs), by using a fixed number m of parameters. Parameters are m additional free second-order variables used in the formulas defining the output, that become: $\phi^i_{\text{pos}}(x, X_1, \ldots, X_m)$, $\phi^i_{\text{lab}_a}(x, X_1, \ldots, X_m)$, $\phi^{i,j}_{\leq}(x, y, X_1, \ldots, X_m)$. "Non-determinism" comes from the fact that the interpretation is obtained after assigning any value to these m second-order variables. In particular an NMSOT may define a non-functional relation.

However, MSOTs, and even NMSOTs, have the *linear-size increase* property: the size of the output word is linearly bounded in the size of the input word. Indeed it is bounded by kn for an input word of length n, where k is the number of copies. This remark is important to distinguish the expressive power of different models.

In Chapter 5, we will also consider *FOT*, the "first-order" fragment of MSOT, i.e. MSOT formulas where no second-order variables occur.

2.2.3 Relations with finite state transducers

Let us now consider the expressive power of MSOTs and NMSOTs, compared to the finite state transducers introduced so far. We refer the reader to Figure 2.1 for an overview of these links.

[8] Note that this definition does not allow to assign an image to the empty word ϵ. An alternative definition allowing it consists in assigning the labels to the *edges* of the logical structure (seen as a graph) [EH01]. We prefer here to assign labels to *nodes* for clarity.

MSOT and two-way transducers The central result here, is the exact correspondence between functional two-way transducers and MSOTs, established by Engelfriet and Hoogeboom [EH01]. This was the first "automata-logic" correspondence at this level for transducers, and renewed the interest of the community for transductions. This is also the main reason why this class is named regular functions.

The proof is constructive in both directions. From a functional two-way transducer, one can build an MSO formula encoding the possible moves in the configuration graph of the transducer, and then build the output word from this, on a given input. The other direction uses an intermediate model of two-way transducers with *MSO jumps*: the transducer can jump from one domain position to another (not necessarily consecutive), but such a jump must be regular, in the sense that the pair of domain positions of the jump must satisfy an MSO formula with two free first-order variables. The proof also uses an extension of MSOT transducers, allowing a regular look-around, which means that the prefix and suffix of the current domain position can be checked against a regular language when trying to apply a transition.

MSOT and one-way transducers A restriction of MSOT capturing exactly functions definable by one-way transducers has been identified in [Boj14] for the origin semantics, and reformulated in [Fil15] in the standard setting.

An MSOT is *order-preserving* if, for every word w in its domain, if $w \models \phi^{i,j}_{\leq}(p, p')$ then $p \leq p'$, i.e. domain position p is to the left of domain position p' in the input word. In other words, when defining the order relation $\leq$ of the output word, one only performs rightward moves in the input word (possibly jumping from one copy to another). A direct construction from order-preserving MSOT to functional one-way transducers, and conversely, is provided in [Fil15]. In Chapter 5, we will use an alternative (but equivalent) definition of order-preserving, closer to one-way transducers.

MSOT and sweeping transducers This correspondence between order-preserving MSOTs and functional one-way transducers can be generalized in order to capture functional sweeping transducers [Bas17]. The corresponding MSOT fragment is called k-phase MSOT and expresses the fact that the copies used by the MSOT can be partitioned into k sets of copies, each set being one-way, i.e. order-preserving, or "order-preserving from right to left".

Formally, an MSOT is *k-phase* if its copies can be partitioned into k sets $C_1, \ldots, C_k$ such that:

- movements inside sets C_h are left-to-right order-preserving if h is odd, and right-to-left order-preserving if h is even. Formally, if $w \models \phi^{i,j}_{\leq}(p, p')$, and copies i and j belong to the same C_h, and h is odd (resp. even), then p is to the left (resp. right) of (or equals) p' in the input word w.
- reversals operate at the extremities of the input word. Hence, if $w \models \phi^{i,j}_{\leq}(p, p')$, and $i \in C_h$ while $j \in C_\ell$, then $\ell = h + 1$ and $p = p'$, and if h is odd (resp. even) then p is the last (resp. first) position of the input word.

Theorem 2.5. *k-phase* MSOTs *exactly capture k-sweeping transducers.*

The details of the construction can be found in [Bas17], and follow the same line as the correspondence between order-preserving MSOTs and functional one-way transducers [Fil15].

MSOT and deterministic transducers To our knowledge, no fragment of MSOT corresponding to sequential functions has been defined.

NMSOT, Hennie machines, and common guess When moving to relations, one could hope that NMSOTs and two-way transducers coincide. This is however not the case, as a two-way transducer may not be of linear-size increase, while all NMSOTs are. For instance the relation that maps a to $\{a^n \mid n \geq 0\}$ is definable by a two-way transducer, but not by an NMSOT.

However, two variants of two-way transducers correspond exactly to NMSOTs. The first one is Hennie machines, as proved in [EH01]. These machines are two-way transducers that can rewrite their input tape (at the reading position), but with the *finite-visit* limitation: each position can be visited only a bounded number of times. In fact, the correspondence also holds in the deterministic case: deterministic Hennie machines with such capabilities correspond to MSOT.

The second model equivalent to NMSOT is deterministic two-way transducers with *common guess* [BDGP17]. The common guess feature consists in annotating the input word with some information from a finite alphabet. Hence a two-way transducer with common guess can perform the transduction that maps a^n to $w\#w$, with $w \in \{a, b\}^*$ and $|w| = n$, while this is impossible without common guess [EH01]. Our definition of NMSOT is based on an assignment of parameters (second-order free variables). This can be simulated by a common guess (annotating the input with the assignment information) followed by an MSOT, and conversely. Thus NMSOTs, and MSOTs with common guess define the same transductions. Now, the proof that MSOTs and functional two-way transducers have the same expressivity also holds when both are preceded by a common guess, so functional two-way transducers with common guess and MSOTs with common guess, and thus NMSOTs, define the same transductions.

2.3 Streaming string transducers

Streaming string transducers [AC10] can be considered as deterministic one-way transducers, enriched with registers. These registers can be used to build parts of the output word.

2.3.1 Definition

A *streaming string transducer* (SST) is a tuple $(Q, \Sigma, \Delta, R, \delta, \nabla, q_0, out)$, where:

- Q is a finite set of states,
- Σ (resp. Δ) is a finite input (resp. output) alphabet,
- R is a finite set of *registers* (distinct from Δ),
- δ is a finite set of transitions, i.e. functions mapping a state and a letter in $Q \times \Sigma$, to an update and a target state in $\nabla \times Q$,
- ∇ is the set of *register updates*, i.e. functions from R to $(R \uplus \Delta)^*$, mapping each register to a word of registers and output letters,
- q_0 is the initial state,
- out is a partial *output function*, mapping some states of Q to a word of registers and output letters in $(R \uplus \Delta)^*$.

During a run, registers will contain some words from Δ^*: we name *valuation* a function $\nu : R \to \Delta^*$ that details the contents of the registers. The *configuration* of an SST is a pair (q, ν) composed by a state and a valuation. The initial configuration is (q_0, ν_0), where ν_0 maps all registers to the empty word ϵ.

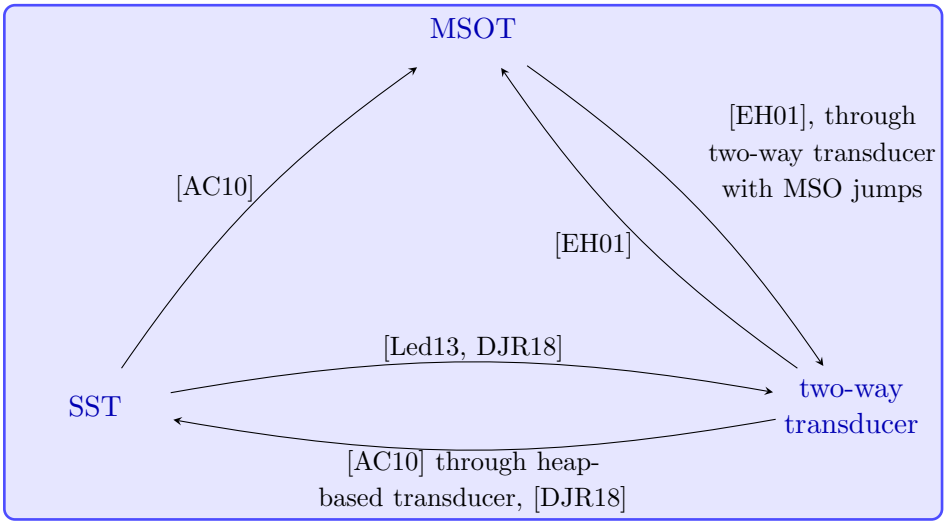


Figure 2.5: Main known translations between models defining regular functions.

To define how configurations are updated, we naturally extend the domain of valuations ν to words in $(R \uplus \Delta)^*$, where each register $r \in R$ is replaced by $\nu(r)$. This way, the word $\nu(up(r))$ will be the content of the register r after applying the update $up \in \nabla$, given a previous valuation ν. Now, given a current configuration (q, ν), if the transducer reads a letter a, then it can update its configuration to (q', ν'), whenever $\delta(q, a) = (up, q')$ and $\nu'(r) = \nu(up(r))$, for every register $r \in R$. We denote this by $(q, \nu) \xrightarrow{a} (q', \nu')$. A *run* of the SST over an input word $u = a_1 \cdots a_n$ is a sequence

$$(q_0, \nu_0) \xrightarrow{a_1} (q_1, \nu_1) \xrightarrow{a_2} \cdots (q_n, \nu_n)$$

It is *successful* if out is defined on q_n, and in this case the output of the transducer is $\nu_n(out(q_n))$.

Note that, by definition, SSTs are deterministic. Their non-deterministic version (*NSST*) is obtained by allowing transition rules δ to be any relation in $Q \times \Sigma \times \nabla \times Q$, not necessarily functional. We will consider NSSTs in Section 4.1.3, where we try to minimize the number of registers.

A key restriction for SSTs is the copyless property. An SST is *copyless* when register contents cannot be duplicated, i.e. each register $r \in R$ appears at most once in the images of all registers by:

(a) an update, i.e. in $up(r_1) \cdots up(r_n)$ where $R = \{r_1, \ldots, r_n\}$, for all $up \in \nabla$, and

(b) the output functions, i.e. in $out(q)$, for every q in the domain of out.

An SST is said *copyful* if it is not copyless. In the whole manuscript, we assume all SSTs to be copyless, unless otherwise stated.

2.3.2 Links with other models

Let us now detail how SSTs relate to the two other models of transformations introduced so far: two-way transducers, and MSOTs.

SST and regular functions The main correspondence is that (copyless) SSTs exactly capture regular functions, i.e. have the same expressiveness as functional two-way transducers and MSOTs. We illustrate some known translations between these three models in Figure 2.5. This has been established in [AC10], where an indirect construction from a deterministic two-way

transducer to an SST is provided (through an intermediate model of heap-based transducer), and also a direct construction from SSTs to MSOTs.

Some other translations have been proposed between the three models. In [DJR18] back-and-forth direct translations between SSTs and deterministic two-way transducers are provided. A prior direct translation from an SST to a two-way transducer has also been proposed in [Led13]. These two translations from SSTs to deterministic two-way transducers are both exponential. A first polynomial translation has been proposed recently [DFJL17], and even builds a reversible two-way transducer.

Capturing rational and sequential functions As SSTs operate in a single left-to-right pass on the input, the link with one-way transducers is easy to establish. An SST is *right-appending* if for all updates $up \in \nabla$ and all registers $r \in R$, the word $up(r)$ is of the form $r' \cdot u$ with $r' \in R$ and $u \in \Delta^*$. This restriction enforces each register to be used as a write-only output tape. Hence:

- one-register right-appending SSTs exactly capture sequential transductions, i.e. those definable by deterministic one-way transducers, and
- right-appending functional NSSTs exactly capture rational transductions, i.e. those definable by functional one-way transducers.

Capturing sweeping transducers Between sequential and rational functions lies the class of functions definable by a sweeping transducer, or equivalently by a k-phase MSOT. This class is also captured by concatenation-free NSSTs: an NSST is *concatenation-free* if each register update contains at most one register name, that is: $up(r) \in \Delta^* \cdot (R \cup \{\epsilon\}) \cdot \Delta^*$ for every update $up \in \nabla$ and every register $r \in R$. This will be proved in Section 4.1.3, where a direct translation between functional sweeping transducers and concatenation-free NSSTs is provided.

Non-determinism One of the nice properties of SSTs, is that the correspondence with MSOT carries over in the non-deterministic case. Hence, NSSTs exactly capture NMSOTs, as shown in [AD11]. As we have seen, this is not the case for two-way transducers. It is also proved in [AD11] that the functionality of an NSST is decidable in PSpace, and that functional NSSTs are equi-expressive to deterministic SSTs.

Copyful SSTs One could wonder why the copyless restriction is added in the definition of SSTs. The reason is to retrieve the linear-size increase property of regular functions, and indeed, as we have seen, this restriction suffices to capture exactly regular functions.

Copyful SSTs can yield outputs of size exponential in the size of the input. Consider for instance the copyful SST with only one register r on the unary alphabets $\Sigma = \Delta = \{a\}$, that performs the updates:

- $r \to a$ when reading the first letter, and then
- $r \to r \cdot r$ when reading the subsequent letters.

This copyful SST implements the function $a^n \to a^{2^{n-1}}$. While being more expressive, copyful SSTs share several interesting properties with their copyless counterpart: functional copyful NSSTs and copyful SSTs have the same expressive power, they have decidable functionality and equivalence problems. Also, it is decidable in PTime whether a copyful SST has an equivalent copyless SST. These results are obtained via back-and-forth translations (in linear time) between copyful SSTs and HDT0L Lindenmayer rewriting systems [FR17].

In [AFT12] it is proved that copyless SSTs have the same expressiveness as *bounded* SSTs where, roughly speaking, an SST is k-bounded if each register content is copied at most k times at any time point of a run. The proof considers infinite strings, but also holds for finite strings. This is studied in further details in [DJR18], where translations between k-bounded SSTs and copyless SSTs are established.

Chapter 3

Two-way to one-way transducers

Contents

This chapter presents two algorithms for the one-way definability of two-way transducers. Both of them take some inspiration from classical proofs for automata.

One of the seminal papers on automata theory is the paper by Rabin and Scott, where the authors investigate several models based on restrictions of Turing machines [RS59]. One of them is two-way automata, the extension of finite state automata where the input head can move in both directions. In that paper, they already prove that two-way automata have the same expressiveness as one-way automata.

Amusingly, an alternative proof of this result has been provided by Shepherdson in the very same volume of the *IBM Journal of Research and Development* [She59]. The story is explained in the paper by Rabin and Scott [RS59]:

> "The result, with its original proof, was presented to the Summer Institute of Symbolic Logic in 1957 at Cornell University. Subsequently J. C. Shepherdson communicated to us a very elegant proof which also appears in this Journal. In view of this we confine ourselves here to sketching the main ideas of our proof."

Indeed the proof by Shepherdson is easier to formalize, even though both proofs can be easily sketched using a graphical representation of two-way runs. Both proofs are described in Section 3.1. We also quickly review some alternative proofs that we did not choose to adapt when moving to transducers.

It may seem surprising that the one-way definability of two-way transducers remained open for such a long time. A lot of results have been obtained for sequential transducers, or rational ones, as discussed in the previous chapter. Moving to regular transductions is challenging, as it involves two-way transducers, which behaviors are more difficult to understand. For instance, as we will see in Chapter 5, their algebraic characterization is wide open. However, the "logic-automata" equivalence obtained by Engelfriet and Hoogeboom [EH01] is a major result in this field, and somehow paved the way towards this one-way definability problem.

The two proofs exposed in this chapter confirm that one has to face many technicalities when dealing with two-way transducers. In this manuscript, we hide many internal lemmas based on combinatorics, and try to give a high-level description, with enough explanations to get the essence of the proofs.

In Section 3.2 we analyze some general properties of two-way transducers and give a first flavor of the arguments we will use in both proofs. Section 3.3 contains our first algorithm for one-way definability, an extension of Rabin-Scott's algorithm to the transducers setting [FGRS13]. Its overall complexity is non-elementary. In Section 3.4 we provide a second algorithm, more related to Shepherdson's proof, yielding an elementary complexity [BGMP18].

Related work. We will review the known constructions proving that two-way automata only recognize regular languages in Section 3.1.3.

One of the most related results is an *iteration lemma* for languages generated by two-way transducers: given a two-way transducer (functional or not), its output language (i.e. the projection, on the output alphabet, of the relation it defines) is k-iterative for some $k \geq 0$. A language is said k-iterative if there exists $N > 0$ such that any output word of length greater than N can be written as $u_1 v_1 \cdots u_k v_k u_{k+1}$ such that $u_1 v_1^n \cdots u_k v_k^n u_{k+1}$ is also in the output language for all $n > 0$. For instance regular languages are 1-iterative, and context-free languages are 2-iterative. Brigitte Rozoy proved that languages generated by two-way transducers are k-iterative [Roz86]. This has also been proved by Tim Smith more recently [Smi14]. The paper by Brigitte Rozoy has some similarities with our Shepherdson-based approach, in particular our definitions of flows and effects. However, the one-way definability problem requires to be more precise than k-iterativity in the analysis of loops. As we will see, we need to "pump" runs in order get equations that imply some periodicity properties, like for iteration lemmas (even though we only need 2 places instead of k). But then we need to "undo pumping" while keeping the periodicity properties, and this part is not needed in iteration lemmas. For this reason we introduce the decomposition of flows into components, for instance, as we will see.

Oblivious two-way automata are two-way automata such that all inputs of the same length have the same shape of runs. Every two-way automaton has an equivalent oblivious one [Pet98], and being oblivious is decidable for deterministic two-way automata [KMP14]. Hence being oblivious may help when comparing two runs of a given automaton (and thus, transducer). But the main constructions presented in this chapter deal with a single run, and its pumped versions. So this would not help us in the present context, but should be kept in mind for others.

In this chapter we also define a family $(f_n)_{n \geq 0}$ of transductions, that can be recognized by a two-way transducer of size polynomial in n, but such that any one-way transducer recognizing it as size at least doubly exponential in n. On the automata side, a tight bound on the gap between two-way and one-way automata has been established by Kapoutsis [Kap05], improving a result of Birget [Bir93], and other prior results [MF71, Bar71, Moo71, SS78, Sip80]. The

precise complexity is known over unary alphabets [KO11]. Also, upper and lower bounds have been recently exhibited for the *shortest word* accepted by a two-way automaton of a given size [DDO19], and for basic operations on two-way automata [JO17, KO12].

3.1 Two-way to one-way automata

In this section we review some translations of two-way automata to one-way automata. We focus on two of them, as they will be the basis for deciding whether a two-way transducer is equivalent to a one-way transducer. The first one is the translation based on crossing sequences proposed by Shepherdson [She59]. The second one is based on the progressive elimination of basic zigzags in the shape of the runs, proposed by Rabin and Scott [RS59].

3.1.1 Crossing sequences: the Shepherdson approach

The easiest translation of two-way automata into one-way automata has been proposed by Shepherdson in 1959. The key idea is to consider, for each position, the sequence of states reached at this position. This sequence, called crossing sequence, can be bounded, and can be used as a state of a one-way transducer. Indeed the transition between two consecutive crossing sequences can be easily checked from the transitions of the two-way transducer.

Normalized runs. A run ρ of a two-way automaton is *normalized* if it does not reaches twice the same state at the same position coming from the same direction, i.e. it does not contain two locations (x, y) and (x, y') where y and y' have the same parity, and $\rho(x, y) = \rho(x, y')$. A two-way automaton is *normalized* if all its runs are.

Any two-way automaton can be normalized. Consider a non-normalized successful run reaching twice the same position x at levels y and y' of same parity, in the same state q. Then the run obtained by removing the part of the run between (x, y) and (x, y') is also a successful run of the same automaton. Moreover, checking that all runs are normalized can be performed easily on the crossing sequences. Therefore, we will always assume that two-way automata are normalized in the sequel.

Crossing sequences. The *crossing sequence* of a run ρ at a given position x is the tuple $(\rho(x, 0), \ldots, \rho(x, h))$ of all the states reached by the run ρ at position x, for the $h + 1$ levels reached at this position. In Figure 3.1, the crossing sequence (q_0, q_2, q_3) is highlighted, for the run of the automaton of Figure 2.4 on the word $\vdash aab \dashv$ at position 3. As we assumed normalized runs, the length of a crossing sequence is bounded by $2|Q| - 1$. We will sometimes refer to the *crossing degree* of a crossing sequence for its length, and the crossing degree of a transducer as the maximal crossing degree of all crossing sequences of its possible runs.

The Shepherdson construction. We now have all the ingredients to define a one-way automaton $\mathcal{A}'$ from a two-way automaton $\mathcal{A}$, following Shepherdson's proof [She59]. The states of $\mathcal{A}'$ are all the possible crossing sequences of size at most $2|Q_{\mathcal{A}}| + 1$ built from the states of $\mathcal{A}$. The transitions of $\mathcal{A}'$ must check that moving from one crossing sequence to the next one is allowed by $\mathcal{A}$.

For consecutive crossing sequences of equal length without internal reversal, one only needs to check that the transition at each level appears in $\mathcal{A}$. This is the case for instance between the crossing sequences at positions 3 and 4 in Figure 3.1. One has also to permit reversals at relevant places, for instance when moving from position 2 to 3, or from 4 to 5.

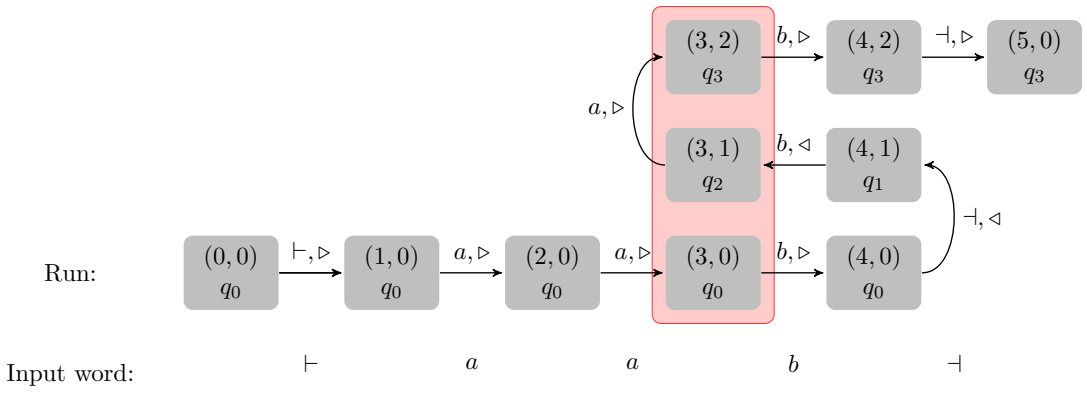


Figure 3.1: Highlighting the crossing sequence (q_0, q_2, q_3) of a run at position 3.

The definition of the transitions of $\mathcal{A}'$ is hence a bit technical but clearly possible, from the transitions of $\mathcal{A}$. We only limit ourselves to intuitions here, but point out a formal proof in Coq of this construction [DS18].

In fact, the exact order on states in crossing sequences is not required: one could keep only two sets: the set of states coming from the left, and the set of states coming from the right. This corresponds to the notion of *frontier* studied in [Bir93, Kap05], that permits to establish tight bounds on the number of states.

3.1.2 Z-motion elimination: the Rabin-Scott approach

Rabin and Scott proposed a totally different approach [RS59].

Z-motions. The shape of a two-way run is arbitrary, and at first sight, there is no easy way to decompose it. Rabin and Scott identified two simple shapes that appear in any two-way run where a reversal occurs [RS59]. They call them z-motions according to these shapes.

A rightward *z-motion* is a part of a run delimited by two positions x_1, x_2 of the input. The z-motion performs a left-to-right pass from x_1 to x_2, then a first reversal at x_2, then a right-to-left pass from x_2 to x_1, then a second reversal at x_1, and finally a left-to-right pass from x_1 to x_2. A rightward z-motion does the symmetric, by starting from the right.

For instance, the upper part of Figure 3.3 shows a run of a two-way automaton $\mathcal{A}$, and two leftward z-motions, between states q_1 and q_2, and between q_3 and q_4 respectively.

The squeeze operation. The key observation is that each z-motion of a two-way automaton $\mathcal{A}$ can be simulated through a single one-way traversal, non-deterministically: The new run maintains the three states of the z-motion in parallel, as depicted in Figure 3.2.

- when entering the z-motion, the state q_1 of the first pass is known, but the state p_5 of the second pass and the state p_6 of the third passes are guessed, and $p_5 \xrightarrow{a,\triangleright} p_6$ must be a transition rule of $\mathcal{A}$;
- during the z-motion simulation, the states of the first and third passes are updated by following the transition rules of $\mathcal{A}$. The state of the second pass is also updated by applying transition rules of $\mathcal{A}$, but in reverse.
- when leaving the z-motion, the states p_2, p_3 of the second and third states must be validated by a transition rule $p_2 \xrightarrow{b,\triangleleft} p_3$ of $\mathcal{A}$.

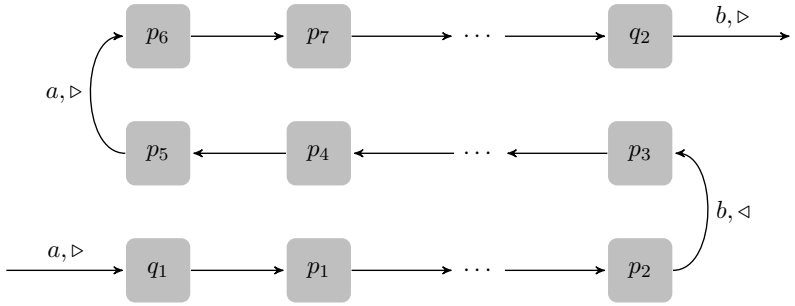


Figure 3.2: Simulating a z-motion via $R^{\mathcal{A}}_{a,b}(q_1, q_2)$.

Let us name $R^{\mathcal{A}}_{a,b}(q_1, q_2)$ the one-way automaton simulating the z-motion beginning in the state q_1 of $\mathcal{A}$ and ending in the state q_2 of $\mathcal{A}$, if the z-motion is rightward and surrounded by a and b letters. We name it $L^{\mathcal{A}}_{a,b}(q_1, q_2)$ if the z-motion it simulates is leftward, and in this case the input is read to the left. This construction is similar to the crossing sequence approach of Shepherdson, but restricted to more basic shapes, and thus easier to define.

We can now define a new automaton, that we name squeeze($\mathcal{A}$), that alternates non-deterministically between two modes:

- in mode $\mathcal{A}$, it just runs $\mathcal{A}$, but with the possibility to guess whether it enters a z-motion, in which case it swaps to mode Z.
- in mode Z (in state q_1), it simulates a z-motion in one pass, by guessing the ending state q_2, and switching to $R^{\mathcal{A}}_{a,b}(q_1, q_2)$ (or $L^{\mathcal{A}}_{a,b}(q_1, q_2)$, depending on the current direction), and guesses at each input letter whether the z-motion ends. If so, it either switches to mode $\mathcal{A}$, or stays in mode Z for a new z-motion.

Global elimination of nested z-motions. The automaton squeeze($\mathcal{A}$) is however limited to the elimination of isolated z-motions, so there is not always a one-way run on every input word. Even worse, a single application of squeeze does not necessarily decrease the degree of nesting of z-motions, as we can see in the first application of squeeze in Figure 3.3.

The original proof only states that "repeating this derivation operation a sufficient number of times, a one-way automaton is obtained which defines the same [language as $\mathcal{A}$]" [RS59], as illustrated in Figure 3.3. In [FGRS13] we proved that it is sufficient to apply squeeze $\boldsymbol{H}^2$ times, where $\boldsymbol{H} = 2|Q| + 1$ is a bound on the length of crossing sequences. Indeed:

- applying squeeze $\boldsymbol{H}$ times decreases the nesting of z-motion sequences by (at least) one,
- and the nesting depth is bounded by $\boldsymbol{H}$.

Hence every run of $\mathcal{A}$ has a corresponding one-way run in squeeze$^{\boldsymbol{H}^2}(\mathcal{A})$. By removing leftward transition rules in squeeze$^{\boldsymbol{H}^2}(\mathcal{A})$, we obtain a one-way automaton recognizing $\mathcal{L}(\mathcal{A})$.

Remark 1. *The nesting structure of z-motions has a direct formulation in* persistent homology*, a domain mostly studied in the computer graphics community [ZC05]. It could be used to associate each z-motion with the factor of the input word it covers (named* persistent interval *in persistent homology), and also inductively on z-motions obtained after successive eliminations. A result of interest is that, if we consider that one step eliminates all elementary z-motions, then the number of steps to eliminate all reversals is bounded by the size of the maximal subset of intersecting intervals. However, in our case, this quantity is unbounded, as observed for instance in a "stairs" shape of run. A deeper understanding of this theory may still have consequences in the field of two-way transducers.*

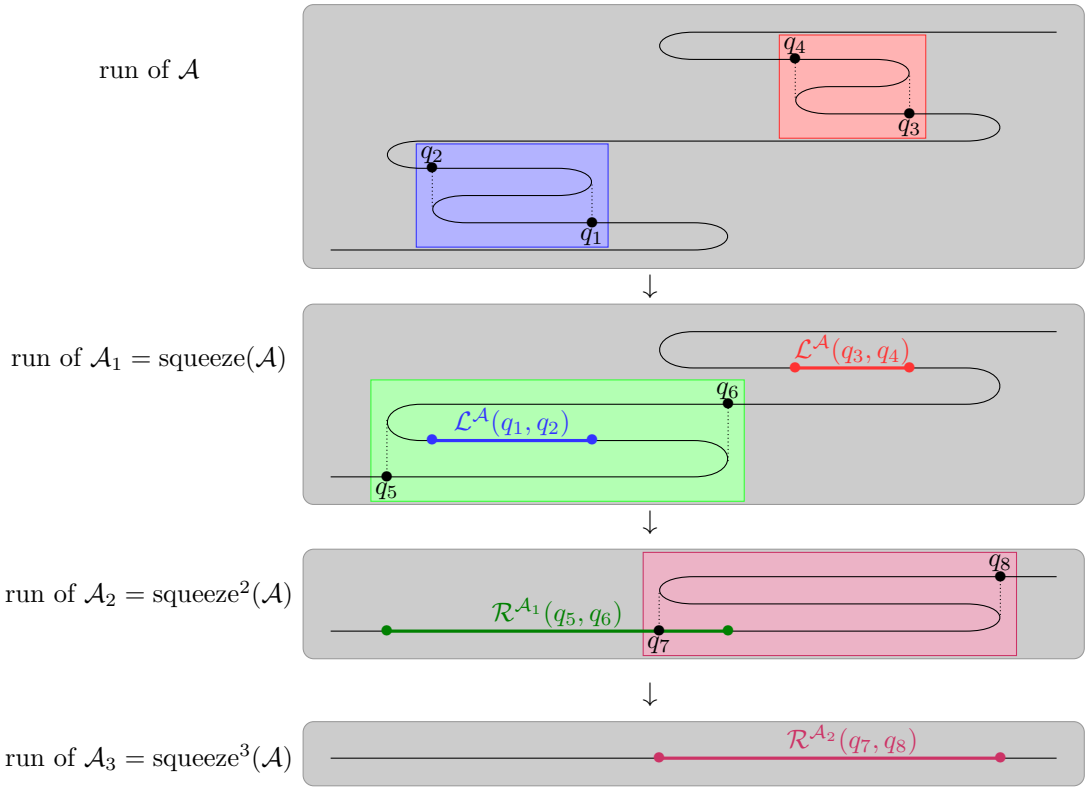


Figure 3.3: Z-motions removal by applications of squeeze.

3.1.3 Other known constructions

For sake of curiosity, we quickly review some other constructions proving that two-way automata can be translated into one-way automata. This section can be safely skipped by the busy reader.

Over-approximation of successful runs. Vardi proposed a kind of "subset construction" on two-way automata, that permits to build a non-deterministic one-way automaton recognizing the *complement* of the language of the two-way automaton [Var89].

In order to estimate if a word u of length n is accepted by a two-way automaton $\mathcal{A} = (Q, \Sigma, \vdash, \dashv, \delta, I, F)$, one can try to build a series of sets $(T_i)_{0 \leq i \leq n+1}$ of states of $\mathcal{A}$, having the following properties:

- T_0 contains the initial states of $\mathcal{A}$: $I \subseteq T_0$,
- T_{n+1} contains no final state of $\mathcal{A}$: $F \cap T_{n+1} = \emptyset$,
- for $0 \leq i \leq n+1$, if $q \in T_i$ and $a \in \Sigma$ then
  - if $(q, a, q', \triangleleft) \in \delta$ then $q' \in T_{i-1}$ (assuming $0 \leq i - 1$)
  - if $(q, a, q', \triangleright) \in \delta$ then $q' \in T_{i+1}$ (assuming $i \leq n$)

This is a kind of over-approximation of the accessibility relation, because it does not take the input into account when crossing a position several times. For instance reading an "a" to the right, followed by reading a "b" to the left is allowed here.

The key observation is that there exists such a sequence (T_i) iff $u \notin \mathcal{L}(\mathcal{A})$. Indeed, if $u \in \mathcal{L}(\mathcal{A})$, i.e. there is a successful run of $\mathcal{A}$, then no such sequence (T_i) exists, as the states

of this run will necessarily appear from $I \subseteq T_0$ to T_{n+1}, and thus $F \cap T_{n+1} \neq \emptyset$. Conversely, if $u \notin \mathcal{L}(\mathcal{A})$, one can build such a sequence (T_i) by putting in each T_i all the states reached at position i in $\mathcal{A}$ on u (assuming $\mathcal{A}$ is complete). This sequence (T_i) verifies all the properties above, in particular $F \cap T_{n+1} = \emptyset$.

Then, one can easily build a non-deterministic one-way automaton building any sequence (T_i) on-the-fly, hence recognizing the complement of $\mathcal{L}(\mathcal{A})$. This automaton has an exponential number of states compared to $\mathcal{A}$. The construction is then refined in order to obtain a deterministic one-way automaton, with $O(2^{|Q|^2})$ states.

Another automata-centric translation has been proposed in [GO14], that transforms any two-way alternating finite automaton into a non-deterministic one-way automaton. This translation is also exponential, and this is tight.

Finite right congruence. Regular languages are characterized by one-way automata, but also by other means, notably algebraic ones. Indeed, regular languages correspond to languages having a finite syntactic monoid, and also to those having a finite right congruence (see Chapter 5). This latter characterization can also be used to show that two-way automata define regular languages, as explained in [Sak09, p. 173], and taught by Schützenberger.

Regular expressions. Yet another characterization of regular languages involves regular expressions. A direct translation of two-way automata into regular expression has been proposed by Hulden [Hul15]. It uses an intermediate word where the input letters are separated by triples (source state, target state, direction). Somehow, these triples describe the crossing sequence arising between positions. Regular expressions are used to check the consistency of the crossing sequence, locally. Then the regular expression is projected in order to recognize the input language.

Logic. Géraud Sénizergues and I are currently writing two new proofs that two-way automata recognize regular languages. Both proofs rely on an idea from Géraud Sénizergues: using "words of words" to encode configurations of two-way automata. The first proof uses the fact that, for order-2 pushdown automata, the set of configurations that such automata can reach, from a regular set of configurations, is itself regular [HO07]. The second one uses logic, and more precisely Muchnik's theorem [Sem84, MS92] to transfer the MSO-definability of a tree-structure $\mathcal{S}^*$ to the MSO-definability of its original relational structure $\mathcal{S}$.

3.2 From automata to transducers

Compared to the automata constructions, deciding the one-way definability of two-way transducers amounts to analyze how the outputs can be produced. In this section we provide some examples, and a high-level analysis of the one-way definability problem. We exhibit some key points that any algorithm for this problem has to address, and some lower bounds.

3.2.1 Properties of two-way transducers: primer

Let us consider a first example, to tackle the limits of one-way definability.

Example 3.1. *We focus on the f_{copy} function, already mentioned in Example 2.2, that maps a word u to uu, over an alphabet $\Sigma = \{a, b\}$.*

If $\text{dom}(f_{\text{copy}}) = \Sigma^*$, then f_{copy} is not one-way definable: the one-way transducer would have to store u, which is impossible with finitely many states.

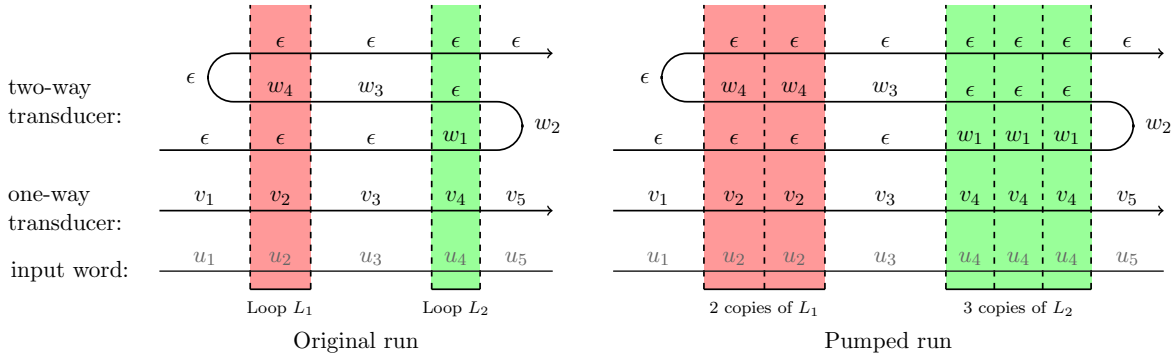


Figure 3.4: Pumping two loops in a simple run.

However, if $\mathrm{dom}(f_{\mathrm{copy}}) = (aba)^*$*, then* f_{copy} *becomes one-way definable: a one-way transducer just needs to output, at each read letter, the next two letters in the sequence* $(aba)^*$*, keeping track of the shift (i.e. emit successively* ab*,* aa *and* ba*).*

This provides a first intuition: there is a tight link between one-way definability, and having outputs with bounded periods inside loops. We will now make this link more explicit.

Loops. When one wants to transform a two-way transducer into a one-way one, the problem is only to deal with "long" outputs. Indeed, "short" (say, bounded) outputs can be guessed and checked, and we could reuse the techniques mentioned for automata. And of course, "long" outputs are due to loops, hence the analysis of loops is the central point here. This is a real difference with automata constructions, where we did not have to consider them at all.

Usual loops in one-way automata are defined as a part of a run that starts and ends in the same state, which allows to build a new run on a new input where the corresponding factors (of the input and the run) are repeated (or "pumped"). When moving to two-way loops, this means that we need to have the same crossing sequences at the borders of the loop, in order to be able to pump both the input and the run. Formally, a *loop* of a run ρ of a two-way transducer over an input word u is an interval of positions $L = [x_1, x_2]$ of u with identical crossing sequences in ρ.

Condition for one-way definability. Roughly speaking, a two-way transducer is not one-way definable when the run order and the input order are somehow *inverted*, i.e. when it has to produce (long) outputs at some position of the input word, but depends on distant (on the right) positions of the input word. This situation will be captured by Property $\mathcal{P}$ in the Rabin-Scott approach, and by the notion of inversion in the Shepherdson approach.

This situation is depicted in Figure 3.4 in the simplest case, i.e. a run that performs 3 passes on the whole input word, and contains two loops L_1 and L_2 that produce some output, for instance $w_1 \neq \epsilon$ and $w_4 \neq \epsilon$. Assume this two-way transducer $\mathcal{T}$ is one-way definable (and that all its runs have this shape). This corresponds to a problematic inversion as described above: the one-way transducer has to output w_4 *after* w_1, but reads u_2 (producing w_4) *before* u_4 (producing w_1). The one-way transducer must also have loops in order to recognize the same domain. For simplicity we assume here that they are delimited by exactly the same positions as the loops of the two-way transducer. Then, when pumping the loop L_1 n_1 times, and the loop L_2 n_2 times, the two transducers must produce the same output, that is:

$$w_1^{n_2} w_2 w_3 w_4^{n_1} = v_1 v_2^{n_1} v_3 v_4^{n_2} v_5$$

The inverted order between n_1 and n_2 in both sides of the equation permits to use some combinatorial tools (that we will expose next). In particular, it ensures that the word $w_1w_2w_3w_4$ (and generally, everything between w_1 and w_4) has a period bounded by some value depending only on the size of the two-way transducer (and not the input word). In other words:

$$\exists u, v, p \text{ such that } \quad w_1w_2w_3w_4 = u^p v \quad \text{and} \quad \begin{cases} |u| \text{ is polynomially bounded in } |\mathcal{T}|, \text{ and} \\ v \text{ is a prefix of } u. \end{cases}$$

This condition (the existence of u, v and p) on every run of $\mathcal{T}$ is thus necessary for one-way definability. It is also sufficient. Indeed, an equivalent one-way run has to guess the positions of the loops (on the fly), the words u and v (there are finitely many), and to output $u^p v$ progressively, checking at each read letter that it outputs the right number of letters (as shown for instance in Example 3.1).

This analysis is limited to runs composed of a single sweep, (and without output in the last pass). The situation is much more involved in the general case:

- in a Rabin-Scott approach (as detailed in Section 3.3), the treatment of a single z-motion will be similar to this analysis, but with the intermediate shape of a z-motion outputting ϵ on the backward pass. One of the difficulties is to prove that a two-way transducer is one-way definable iff all its z-motions are (inductively). The squeeze operation and the construction of the one-way transducer will be very similar to that automata case.
- in a Shepherdson approach (as detailed in Section 3.4), two difficulties arise:
  1. with multiple sweeps, there can be several pairs of one-way definable loops like (L_1, L_2), but at different levels. As we will see, they have to form "stairs". This will induce a notion of "block decomposition".
  2. generally, reversals may appear at inner positions, not only at the border, and in particular inside loops. It becomes difficult to derive word equations that will ensure bounded periodicity. We will need to identify components inside loops, and focus on idempotent loops to circumvent this.

Combinatorics toolbox. Most of the combinatorial parts of the proofs are solved using Fine-Wilf's theorem.

Theorem 3.1 (Fine-Wilf's theorem [FW65])**.** *If $w_1 = w_1' \, w \, w_1''$ has period p_1, $w_2 = w_2' \, w \, w_2''$ has period p_2, and the common factor w has length at least $p_1 + p_2 - \gcd(p_1, p_2)$, then w_1, w_2, and $w_3 = w_1' \, w \, w_2''$ have period $\gcd(p_1, p_2)$.*

From Fine-Wilf's theorem, we derive two properties. The first one gives, from the equation obtained from pumping loops, a bound on the period of the pumped output.

Lemma 3.1. *Consider a word equation of the form*

$$v_0^{(n_1,n_2)} \, \boldsymbol{v_1^{n_1}} \, v_2^{(n_1,n_2)} \, \boldsymbol{v_3^{n_2}} \, v_4^{(n_1,n_2)} \;=\; w_0 \, \boldsymbol{w_1^{n_2}} \, w_2 \, \boldsymbol{w_3^{n_1}} \, w_4$$

where n_1, n_2 are the unknowns, v_1, v_3 are non-empty words, and $v_0^{(n_1,n_2)}$ is a word with iterated factors of the form $v_0^{n_1}$ or $v_0^{n_2}$ (and resp. for $v_2^{(n_1,n_2)}$ and $v_4^{(n_1,n_2)}$). If the above equation holds for all $n_1, n_2 \in \mathbb{N}$, then

$$\boldsymbol{v_1} \; \boldsymbol{v_1^{n_1}} \; v_2^{(n_1,n_2)} \; \boldsymbol{v_3^{n_2}} \; \boldsymbol{v_3}$$

has period $\gcd(|v_1|, |v_3|)$ for all $n_1, n_2 \in \mathbb{N}$.

The second property allows to transfer the properties from the pumped run, to the original run.

Lemma 3.2. *Assume that $v_0 \, \boldsymbol{v_1^n} \, v_2 \cdots v_{k-1} \, \boldsymbol{v_k^n} \, v_{k+1}$ has period p for some $n > p$. Then $v_0 \, \boldsymbol{v_1}^{\boldsymbol{n_1}} \, v_2 \cdots v_{k-1} \, \boldsymbol{v_k}^{\boldsymbol{n_k}} \, v_{k+1}$ has period p for all $n_1, \ldots, n_k \in \mathbb{N}$.*

Subruns. A difference between automata and transducers, is that, in the case of transducers, there may exist parts of runs that are not one-way definable even if the transducer is one-way definable. So, we will always take care of considering only parts of runs that can be embedded in a *successful* run. This can be enforced in all the proofs exposed here, but we skip the details for clarity.

Normalized runs. Still, some properties on runs of two-way automata transfer to two-way transducers. Notably, functional two-way transducers can be assumed to be normalized, in the exact same manner as for two-way automata, but for a different reason. Consider a non-normalized run, and a crossing sequence with two locations (x, y) and (x, y') associated with the same state, with y and y' of same parity. Then the part of the run between these two locations must have the empty word as output, otherwise another output word could be produced on the same input word by repeating this part, which would contradict functionality. The run obtained by removing this part is also successful (with the same output), and we can discard the non-normalized run. As for automata, this can be check easily on crossing sequences.

3.2.2 Lower bounds for one-way definability

Before describing some algorithms for deciding one-way definability, let us exhibit some lower bounds for this problem. The first one is obtained by encoding the emptiness of the intersection of a set of deterministic finite state automata, which is PSpace-hard.

Proposition 3.1. *One-way definability of deterministic two-way transducers is* PSpace*-hard.*

If one wants to *build* the equivalent one-way transducer, it will be of doubly exponential size in the worst case.

Proposition 3.2. *There exists a family $(f_n)_{n\in\mathbb{N}}$ of functions from $\{0,1\}^*$ to $\{0,1\}^*$ such that:*

- *every f_n can be implemented by a sweeping transducer of size $O(n^2)$, and*
- *every f_n is one-way definable, but*
- *every one-way transducer implementing f_n has at least $\Omega(2^{2^n})$ states.*

Consider for instance the "copy" function $f_n(u) = uu$, but on specific domains:

$$\mathrm{dom}(f_n) = \{a_0 w_0 \cdots a_{2^n-1} w_{2^n-1} \mid \forall i, a_i \in \{0,1\} \text{ and } w_i \text{ is the binary encoding of } i \text{ on } n \text{ bits}\}$$

In other words, $w_0 = 0^n$, $w_1 = 0^{n-1}1$, ..., and $w_{2^n-1} = 1^n$. A sweeping transducer of size $O(n^2)$ can implement f_n. This transducer uses its first n sweeps to check the w_i's: the jth sweep checks the jth bits to the right of w_i, using the $1 \to 0$ changes in the $(j+1)$th bit as a hint to change its bit. Then, two additional sweeps are used to copy the input.

It is also possible to implement each f_n by a one-way transducer, by simply outputting the a_i's and storing them, and also storing one w_i at a time to check their correctness. It can be shown that any one-way transducer implementing f_n has to store a word of exponential size, and thus requires a doubly exponential number of states [BGMP18].

3.3 Rabin-Scott approach

Let us now adapt the Rabin-Scott approach to transducers [FGRS13]. From a functional two-way transducer $\mathcal{T}$, we first define its z-motion transductions $\mathcal{R}^{\mathcal{T}}_{a,b}(q_1, q_2)$ and $\mathcal{L}^{\mathcal{T}}_{a,b}(q_1, q_2)$, similarly to $\mathcal{R}^{\mathcal{A}}_{a,b}(q_1, q_2)$ and $\mathcal{L}^{\mathcal{A}}_{a,b}(q_1, q_2)$ for the automaton $\mathcal{A}$. Then we exhibit a necessary and

sufficient condition for a z-motion transduction to be one-way definable. This Property $\mathcal{P}$ is central to our proof. It consists in a combinatorial condition on all loops of a z-motion transducer, but we prove that is is semantical, i.e. every z-motion transducer $\mathcal{Z}'$ equivalent to $\mathcal{Z}$ also verifies Property $\mathcal{P}$. Then we show that if $\mathcal{T}$ is one-way definable, then all z-motion transductions $\mathcal{R}^{\mathcal{T}}_{a,b}(q_1, q_2)$ and $\mathcal{L}^{\mathcal{T}}_{a,b}(q_1, q_2)$ also are, which permits to use the squeeze operation on $\mathcal{T}$ exactly like we did on automata, and obtain an equivalent transducer with some removed z-motions. This yields a decision procedure, by successively applying squeeze, each time checking that Property $\mathcal{P}$ holds on all z-motion transducers of the current transducer. If it fails before $\boldsymbol{H}^2$ applications of squeeze, then $\mathcal{T}$ is not one-way definable. Otherwise it is, and $\text{squeeze}^{\boldsymbol{H}^2}(\mathcal{T})$ is one-way, once its leftward transitions are removed. This section describes these steps.

3.3.1 Decomposing into elementary z-motions

Z-motion transducers. A *z-motion transducer* is similar to a two-way transducer, but its successful runs must end in a final state *and* have a rightward z-motion shape, as depicted in Figure 3.2.[1] Also, the input word is not necessarily surrounded by $\vdash, \dashv$ letters, as z-motion transducers will be defined from subruns of two-way transducers. A *z-motion transduction* is a transduction that can be associated with a z-motion transducer. While z-motion transducers are not strictly speaking transducers, we use the same terminology, and most of the definitions apply.

Given a two-way transducer $\mathcal{T}$, we define its z-motion transductions $\mathcal{L}^{\mathcal{T}}_{a,b}(q_1, q_2)$ (resp. $\mathcal{R}^{\mathcal{T}}_{a,b}(q_1, q_2)$), where $a, b \in \Sigma$ and q_1, q_2 are states of $\mathcal{T}$, as the set of pairs (u, v) obtained by considering a part of a successful run of $\mathcal{T}$ operating on the factor u of the input word, having a shape of leftward (resp. rightward) z-motion, outputting v, starting in q_1 and ending in q_2.[2]

Property $\mathcal{P}$. Central to our proof is a characterization of one-way definable z-motion transductions by the following property.

Definition 3.1 (Property $\mathcal{P}$)**.** *Let $\mathcal{Z}$ be a z-motion transducer. We say that $\mathcal{Z}$ satisfies the property $\mathcal{P}$ if for all words $u \in \text{dom}(\mathcal{Z})$, for all successful runs ρ on u, and for all pairs of loops (i_1, j_1) and (i_2, j_2) of ρ such that $j_1 \leq i_2$, there exist $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5 \in \Sigma^*$, $f, g : \mathbb{N}^2 \to \Sigma^*$ and constants $c_1, c'_1, c_2, c'_2 \geq 0$ such that $c_1, c_2 \neq 0$ and for all $k_1, k_2 \geq 0$,*

$$f(k_1, k_2)x_0 v_1^{\eta_1} x_1 w_1^{\eta_2} x_2 w_2^{\eta_2} x_3 v_2^{\eta_1} x_4 v_3^{\eta_1} x_5 w_3^{\eta_2} x_6 g(k_1, k_2) = \beta_1 \beta_2^{k_1} \beta_3 \beta_4^{k_2} \beta_5$$

where $\eta_i = k_i c_i + c'_i$, $i \in \{1, 2\}$, and, x_i's, v_i's and w'_is are words defined as depicted in Figure 3.5.

Let us briefly give some intuitions behind this word equation. The property will be used to characterize one-way definability of z-motion transducers, so imagine an equivalent one-way transducer running in parallel, on the same input u. When pumping the loops of the two-way transducer, the one-way transducer will have to produce the same output. So there must be corresponding loops in the one-way transducer. However, they may appear with some shift, and several occurrences of one loop may correspond to several occurrences of the corresponding loop. This explains that η_i and k_i may differ, and are linearly related. This also explains the functions f and g: when pumping, parts that are outside the z-motion may be produced differently in the two transducers.

[1] The "semantical" restriction on the shape of successful runs is "syntactically" checked when they are removed, i.e. simulated in a one-way manner, as described later.

[2] More precisely, in order for $\mathcal{L}^{\mathcal{T}}_{a,b}(q_1, q_2)$ to be rightward, we actually define it on the mirror of $\mathcal{T}$. This makes no significant difference in the proofs.

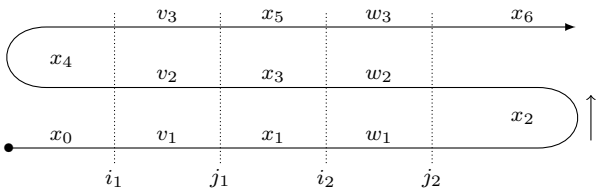


Figure 3.5: Output decomposition in property $\mathcal{P}$.

Besides these technicalities, the main point here is the alternating c_1 / c_2 / c_2 / c_1 on the left side (through k_i's), versus c_1 / c_2 on the right side. This will allow to infer some periodicity property on the output word, and thus a way to produce it from left to right.

As expected, Property $\mathcal{P}$ characterizes one-way definable z-motion transducers:

Proposition 3.3. *A z-motion transducer $\mathcal{Z}$ is one-way definable iff it satisfies Property $\mathcal{P}$, and in this case, a corresponding one-way transducer can be built. Moreover, the Property $\mathcal{P}$ is decidable.*

The proof of this result is deferred to Section 3.3.3. We focus now on the decision algorithm itself: from this characterization of z-motions, how to obtain a characterization for the whole transducer?

3.3.2 Decision algorithm

Squeeze operator. In Section 3.1.2, we defined squeeze($\mathcal{A}$) as the two-way automaton built from the two-way automaton $\mathcal{A}$, where some z-motions (selected non-deterministically) are removed. In the case of transducers, this will not always be possible. However, if a transducer $\mathcal{T}$ is known to be one-way definable, then we will see that squeeze(T) can be defined. This is maid possible by the following fact.

Proposition 3.4. *If $\mathcal{T}$ if one-way definable, then all its z-motion transductions $\mathcal{L}^{\mathcal{T}}_{a,b}(q_1, q_2)$ and $\mathcal{R}^{\mathcal{T}}_{a,b}(q_1, q_2)$ are also one-way definable.*

This is proved by showing that Property $\mathcal{P}$ holds in this case, and by applying Proposition 3.3. The proof is not complex, and follows the explanations following the definition of the Property $\mathcal{P}$.

Hence, if $\mathcal{T}$ is one-way definable, then squeeze($\mathcal{T}$) is defined in the same manner as for automata (see Section 3.1.2), the only real difference is how a z-motion is replaced by a one-way part of run: This will be detailed in Section 3.3.3.

Algorithm. Let us now explicit in Algorithm 1 the decision algorithm for deciding whether a functional two-way transducer $\mathcal{T}$ is one-way definable. This algorithm tries to apply $\boldsymbol{H}^2$ times the squeeze operator on $\mathcal{T}$. Each time, it checks whether all its z-motion transductions are one-way definable. Indeed, if at some point this test fails, then we have found a transducer equivalent to $\mathcal{T}$, that is not one-way definable (by Proposition 3.4), and thus $\mathcal{T}$ itself is not one-way definable. Otherwise, after $\boldsymbol{H}^2$ applications of squeeze, one gets a transducer equivalent to $\mathcal{T}$, that is one-way once leftward transitions are removed (for the same reason as for automata, see Section 3.1.2), and thus $\mathcal{T}$ is one-way definable.

Testing whether all z-motion transductions of a transducer are one-way definable is decidable: One can define all the z-motion transducers recognizing them, and then check Property $\mathcal{P}$ on them (using Proposition 3.3).

Algorithm 1 Deciding one-way definability of a functional two-way transducer, by z-motion elimination.

```
1: function OneWayDefinable(functional two-way transducer 𝒯) : Boolean
2:     i ← 0
3:     𝒯′ ← 𝒯
4:     while all z-motion transductions of 𝒯′ are one-way definable and i ≤ H² do
5:         𝒯′ ← squeeze(𝒯′)
6:         i ← i + 1
       return i > H²
```

Complexity. The time complexity of Algorithm 1 is a tower of exponentials whose height depends on the size of $\mathcal{T}$, and is thus non elementary. Indeed, the squeeze operator produces a transducer that is doubly exponential in the size of its input transducer, as we will see later: it is decomposed in two successive steps, both of which induce an exponential blowup as they need to guess a word of size polynomial in the size of their input transducer. As squeeze is applied $\boldsymbol{H} = 2|Q| + 1$ times in the worst case, we obtain such a tower.

3.3.3 Dealing with elementary z-motions

In this Section we prove Proposition 3.3, i.e. that for z-motion transducers, Property $\mathcal{P}$ characterizes one-way definability, and is decidable.

Property $\mathcal{P}$ is semantical and necessary. Property $\mathcal{P}$ describes a property of the loops of a transducer, and as such it seems related to it. But in fact it is related to the transduction itself. This will allow us to reuse it at any time point during the transformation of the transducer.

Proposition 3.5. *Let $\mathcal{T}$ and $\mathcal{T}'$ be two equivalent z-motion transducers. $\mathcal{T}$ satisfies Property $\mathcal{P}$ iff $\mathcal{T}'$ also does.*

A first consequence is that Property $\mathcal{P}$ is a necessary condition for a z-motion transducer $\mathcal{T}$ to be one-way definable. Indeed, if $\mathcal{T}$ is one-way definable, we can take an equivalent one-way transducer and turn it into a z-motion transducer (by adding two passes producing ϵ). It can easily be checked that this latter z-motion transducer satisfies Property $\mathcal{P}$, and by Proposition 3.5, so does $\mathcal{T}$.

Lemma 3.3. *If a z-motion transducer is one-way definable, then it satisfies Property $\mathcal{P}$.*

Property $\mathcal{P}$ implies one-way definability. Proving the converse of Lemma 3.3 requires to exploit Property $\mathcal{P}$ in order to turn a z-motion transducer into a one-way transducer. This constitutes the most technical part of the proof.

In order to simplify the proof, in particular the word combinatorics, we proceed in two steps, by introducing ϵ-z-motion transducers as intermediate model. An *ϵ-z-motion transducer* is simply a z-motion transducer producing ϵ in its backward pass. The structure of the proof is then:

1. we define Property $\mathcal{P}_1$ (describing the periodicity of the outputs in loops), show that it is implied by Property $\mathcal{P}$, and that any z-motion transducer satisfying Property $\mathcal{P}_1$ can be transformed into an equivalent ϵ-z-motion transducer.

2. similarly, we define Property $\mathcal{P}_2$ of ϵ-z-motion transducers, show that it is also implied by Property $\mathcal{P}$, and that any ϵ-z-motion transducer satisfying Property $\mathcal{P}_2$ can be transformed into an equivalent one-way transducer.

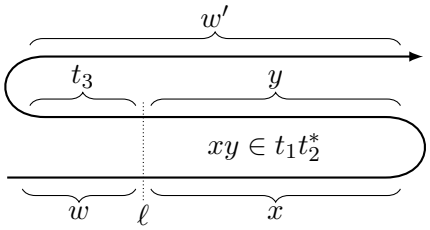


Figure 3.6: Decomposition of the output according to Property $\mathcal{P}_1$.

We only sketch the first step in this document, the second one follows the same ideas. It is interesting to explain this step, because it shows how to shift from a property on the existence of loops (Property $\mathcal{P}$) to a property on the periodicity of the output (Property $\mathcal{P}_1$). Property $\mathcal{P}_1$ is depicted in Figure 3.6.

Definition 3.2 (Property $\mathcal{P}_1$)**.** *Given a z-motion transducer $\mathcal{Z}$, and a pair $(u, v) \in [\![\mathcal{T}]\!]$, we say that (u, v) satisfies Property $\mathcal{P}_1$ if for all runs ρ of $\mathcal{Z}$ on u, there exists a position ℓ of u and words $w, w', t_1, t_2, t_3 \in \Delta^*$ such that:*

- *w (resp. x) is the output of ρ on the first pass, before (resp. after) ℓ, and*
- *y (resp. t_3) is the output of ρ on the second pass, on the right of (resp. left of) ℓ, and*
- *w' is the output of ρ on the third pass, and*
- *$xy \in t_1 t_2^*$, and*
- *$|t_i| \leq 4.o.m^3.|\Delta|$ for every $i \in \{1, 2, 3\}$, where m is the number of states of $\mathcal{Z}$ and o the size of the longest word in its transition rules.*

We say that $\mathcal{Z}$ satisfies Property $\mathcal{P}_1$ if all $(u, v) \in [\![\mathcal{Z}]\!]$ satisfy it.

The important point in the definition is the last one, i.e. words t_i are "small", i.e. bounded in the size of the transducer, independently of the word. This permits to build an equivalent ϵ-z-motion transducer.

Proposition 3.6. *If a z-motion transducer satisfies Property $\mathcal{P}_1$, then one can build an equivalent ϵ-z-motion transducer.*

The goal here is to produce the outputs of the first and second pass of the z-motion transducer $\mathcal{Z}$ only during the first pass. This can be obtained by guessing the t_i's, and also guessing the position ℓ. Before ℓ, w is output, and t_3 is checked (running transitions in backward). After ℓ, it remains to produce xy, i.e. the word in $t_1 t_2^*$ with the same length as xy. This is achieved by outputting a prefix of $t_1 t_2^*$ progressively. Each time a letter of the input word is read, it outputs the same amount of letters of $t_1 t_2^*$ as the transitions used by $\mathcal{Z}$ on the first and second pass, and a counter is used to keep track of the current position in $t_1 t_2^*$ (this counter is bounded: it does not need to distinguish the copies of t_2). Then, at the end of the first pass, t_3 is output, and the second pass can be performed without outputting anything.

Property $\mathcal{P}_2$ is similar but on ϵ-z-motion transducers: it states that one can find a position ℓ_1 in the first pass, and a position ℓ_2 in the third pass, such that the output between ℓ_1 and ℓ_2 is of the form $t_1 t_2^* t_3$, which permits the construction of an equivalent one-way transducer with the same kind of technique.

Moreover, Property $\mathcal{P}$ implies Property $\mathcal{P}_1$ and Property $\mathcal{P}_2$. These proofs rely on word combinatorics. They consist in case analyses, some of them being solved using Fine-Wilf's theorem (Theorem 3.1).

Proof of Proposition 3.3. We can now plug these ingredients to get a proof that a z-motion transducer $\mathcal{T}$ is one-way definable iff it satisfies Property $\mathcal{P}$. By Lemma 3.3, if $\mathcal{Z}$ is one-way definable, then it satisfies Property $\mathcal{P}$.

For the converse, assume now that $\mathcal{T}$ satisfies Property $\mathcal{P}$. As we have seen, this implies that $\mathcal{T}$ also satisfies Property $\mathcal{P}_1$, and we can, by Proposition 3.6, build an equivalent ϵ-z-motion transducer $\mathcal{Z}'$. By Proposition 3.5, $\mathcal{Z}'$ also satisfies Property $\mathcal{P}$, and thus also Property $\mathcal{P}_2$. This allows us to build an equivalent one-way transducer $\mathcal{Z}''$.

It remains to show that Property $\mathcal{P}$ is decidable. It suffices to build $\mathcal{Z}''$ as explained above, and then check whether $\mathcal{Z}''$ is equivalent to $\mathcal{Z}$. Indeed, if $\mathcal{Z}$ is one-way definable, they will be equivalent. If $\mathcal{Z}$ is not one-way definable, then they cannot be equivalent, as $\mathcal{Z}''$ is one-way.

3.4 Shepherdson approach

As we have seen, the Rabin-Scott approach for deciding one-way definability led to an algorithm with non-elementary complexity. We study now the Shepherdson approach with the objective of improving the complexity. The price to pay, is that the proof will be less "compositional", and we will have to carefully analyze the loops of two-way transducers, a challenging task.

3.4.1 Results and road map

The Shepherdson approach leads to the following result, that we prove in this section.

Theorem 3.2. *There is an algorithm that takes as input a functional two-way transducer $\mathcal{T}$ and outputs in* 3ExpTime *a one-way transducer $\mathcal{T}'$ satisfying the following properties:*

1. $\mathcal{T}' \subseteq \mathcal{T}$,
2. $\mathrm{dom}(\mathcal{T}') = \mathrm{dom}(\mathcal{T})$ *if and only if $\mathcal{T}$ is one-way definable,*
3. $\mathrm{dom}(\mathcal{T}') = \mathrm{dom}(\mathcal{T})$ *can be checked in* 2ExpSpace.

Moreover, if $\mathcal{T}$ is a sweeping transducer, then $\mathcal{T}'$ can be constructed in 2ExpTime *and* $\mathrm{dom}(\mathcal{T}') = \mathrm{dom}(\mathcal{T})$ *is decidable in* ExpSpace.

In order to prove this theorem, we first focus on the conditions for one-way definability. We already gave some intuitions on the notions of inversion and decomposition. These will be explained in further details later.

Theorem 3.3. *Given a functional two-way transducer $\mathcal{T}$, an integer $\boldsymbol{B}$ can be computed such that the following are equivalent:*

P1) *$\mathcal{T}$ is one-way definable,*

P2) *for every successful run of $\mathcal{T}$ and every inversion in it, the output produced amid the inversion has period at most $\boldsymbol{B}$,*

P3) *every successful run of $\mathcal{T}$ admits a $\boldsymbol{B}$-decomposition.*

We first prove **P1** $\to$ **P2** $\to$ **P3** for sweeping transducers, which loops are easier to analyze. We then analyze the general two-way case, and prove **P1** $\to$ **P2** $\to$ **P3** $\to$ **P1** for two-way transducers. Indeed **P3** $\to$ **P1** is not simpler in the sweeping case. The proof that Theorem 3.3 implies Theorem 3.2 is deferred to the end of this Section.

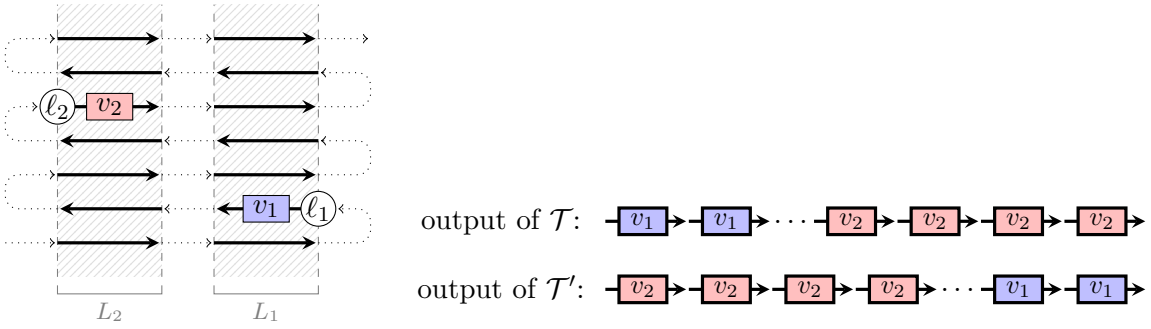


Figure 3.7: An inversion and the effect of pumping in an equivalent one-way transducer $\mathcal{T}'$.

3.4.2 Sweeping case

From now on, we fix a functional non-deterministic sweeping transducer $\mathcal{T}$ and analyze one of its successful runs ρ.

Inversions: P1 → P2. The notion of inversion is illustrated in Figure 3.7.

Definition 3.3 (Inversion (sweeping case))**.** *An inversion of the run ρ is a tuple $(L_1, \ell_1, L_2, \ell_2)$ such that*

1. *L_1, L_2 are loops of ρ,*
2. *$\ell_1 = (x_1, y_1)$ and $\ell_2 = (x_2, y_2)$ are the first positions of factors of ρ in L_1 and L_2, respectively,*
3. *$x_1 > x_2$, while ℓ_2 follows ℓ_1 in the run,*
4. *for both $i = 1$ and $i = 2$, $\mathsf{out}(\mathsf{tr}(\ell_i))$ is non-empty (where $\mathsf{tr}(\ell_i)$ is the part of the run starting at ℓ_i and crossing L_i) and*
5. *there is no loop strictly included in L_i producing a non-empty output at the level of $\mathsf{tr}(\ell_i)$ (for both $i \in \{1, 2\}$).*

This definition formalizes the intuitions given in Section 3.2.1: an inversion between the run order and the input order (Condition 3), producing non-empty outputs (Condition 4).

Condition 5 requires that these inversions are "minimal", in the sense that they do not include loops with the same property. This allows to bound the outputs v_1 and v_2 of both factors of ρ in L_1 (resp. L_2) starting at ℓ_1 (resp. L_2) by the constant $\boldsymbol{B} = \boldsymbol{C}|Q|^{\boldsymbol{H}} + 1$ where $\boldsymbol{C}$ is the maximal length of outputs in the transitions of $\mathcal{T}$, and $\boldsymbol{H} = 2|Q| - 1$ is the maximal length of a crossing sequence. This will be useful for bounding the period, as it will be bounded by $|v_1|$ and $|v_2|$, and thus by $\boldsymbol{B}$. Indeed, we can prove the following property, i.e. **P1** → **P2** (a slightly stronger statement is used in the complete proof):

Proposition 3.7. *If the functional sweeping transducer $\mathcal{T}$ is one-way definable, then the following property* **P2** *holds:*

For all inversions $(L_1, \ell_1, L_2, \ell_2)$ of ρ, the period of $\mathsf{out}(\rho[\ell_1, \ell_2])$ is bounded by $\boldsymbol{B}$.

The proof follows the ideas exposed on a simple case in Section 3.2.1. It uses the combinatorial properties based on Fine-Wilf's theorem: Lemma 3.1 to bound the period of the output in the pumped run, and Lemma 3.2 to lift it to the original run.

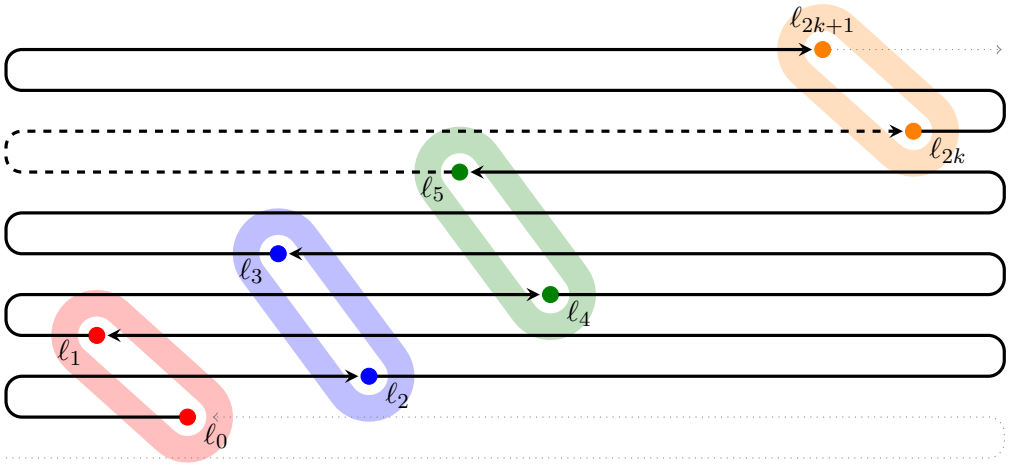


Figure 3.8: A non-singleton S^*-equivalence class seen as a series of overlapping inversions.

Run decomposition: P2 → P3. From this property **P2**, we aim now at building a complete decomposition of the run ρ into "one-way definable" parts. A first step towards such a decomposition, is to generalize Proposition 3.7 to a series of overlapping inversions, instead of a single one.

We formalize this through the binary relation S. For every pair of locations ℓ, ℓ' of ρ, we have $\ell \mathsf{S} \ell'$ iff ℓ and ℓ' are inside the same inversion $(L_1, \ell_1, L_2, \ell_2)$ of ρ, i.e. between ℓ_1 and ℓ_2. We denote by S^* the reflexive and transitive closure of S, which constitutes an equivalence relation. The relation S^* may gather distinct inversions together, and in fact it gathers *overlapping* inversions, in the intuitive way: the second inversion begins inside the first one. Consider for instance the situation depicted in Figure 3.8. The inversions $(L_0, \ell_0, L_1, \ell_1)$ and $(L_2, \ell_2, L_3, \ell_3)$ overlap, as ℓ_2 is between ℓ_0 and ℓ_1. As stated above, this permits to extend the bounded periodicity of the output to a whole equivalence class of S^*:

Lemma 3.4. *If ρ satisfies* **P2** *and if ℓ and ℓ' are two locations of ρ such that ℓ precedes ℓ' in ρ, and $\ell \mathsf{S}^* \ell'$, then $\mathsf{out}(\rho[\ell, \ell'])$ has period at most $\boldsymbol{B}$.*

However, S^* may have several equivalence classes. This corresponds to having some series of inversions that do not overlap, typically on separate levels. Indeed we need a more global notion of "decomposition" of ρ, as depicted in Figure 3.9. The thick arrows indicate outputs that are unbounded, but with a bounded period, while dotted lines denote parts with a bounded output. Let us define a $\boldsymbol{B}$-decomposition of ρ and show how it applies on the depicted run.

A $\boldsymbol{B}$-*decomposition* of ρ is a partition of ρ into $\boldsymbol{B}$-blocks and $\boldsymbol{B}$-diagonals, where:

- a $\boldsymbol{B}$-block is a factor $\rho[\ell, \ell']$ of ρ (like B_1 and B_2) which output has a period bounded[3] by $2\boldsymbol{B}$, and such that the output produced in positions between $\ell = (x, y)$ and $\ell' = (x', y')$ but outside $[x, x']$ (depicted as dotted lines) is bounded by $2\boldsymbol{HB}$.

- a $\boldsymbol{B}$-diagonal is a factor $\rho[\ell, \ell']$ of ρ (like D_1 and D_2) made of floors (thick arrows in the figure), i.e. rightward factors of ρ, that do not overlap vertically and that have a period bounded by $2\boldsymbol{B}$, and separated by factors of ρ (dotted lines) having an output bounded by $2\boldsymbol{HB}$.

[3] strictly speaking, we allow here a prefix and a suffix of *length* at most $2\boldsymbol{B}$, and the word in-between has *period* at most $2\boldsymbol{B}$.

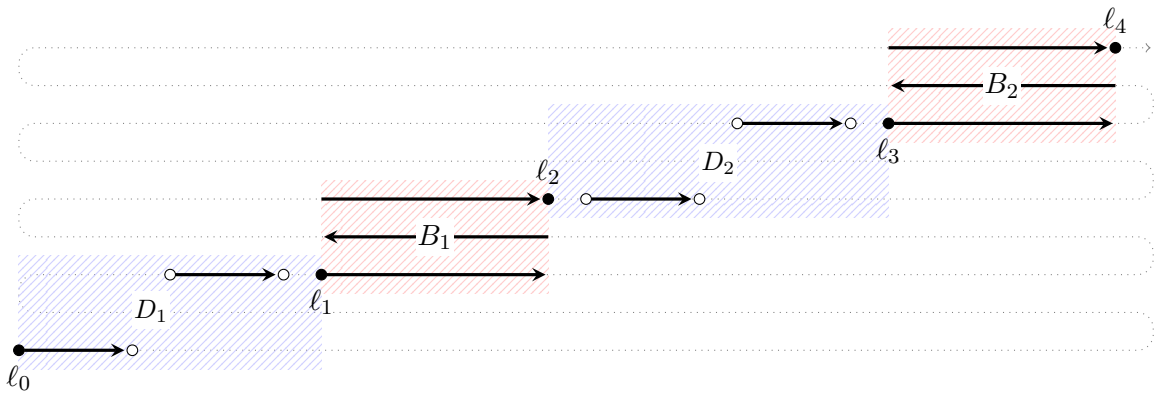


Figure 3.9: Decomposition of a run into diagonals and blocks.

Now, it remains to show that Property **P2** implies a $\boldsymbol{B}$-decomposition. We already did a part of the job when studying the overlapping inversions (through the relation S^*). Indeed, we can show that:

1. every (non-singleton) equivalence class of S^* defines a $\boldsymbol{B}$-block,
2. outside these blocks, every part of the run is a $\boldsymbol{B}$-diagonal.

For the first point, we do not explicit in details how a $\boldsymbol{B}$-block is "defined" from an equivalence class of S^*, but, roughly speaking, it is the bounding box including all inversions of that class. This ensures that, on the left and on the right of a $\boldsymbol{B}$-block (at the same levels), the output is bounded: otherwise, there would be an inversion in this area, which should have been included in that bounding box. The bound on the period of $\boldsymbol{B}$-blocks is given by Lemma 3.4.

The second point is proved in two steps. First, one can show that two distinct $\boldsymbol{B}$-blocks cannot overlap vertically (i.e. share some positions of the input word): they would have to merge. Now, between two blocks, productive loops have to be arranged in a monotonic way (i.e. a loop has to use input positions after the preceding one, and produce only at one level), otherwise they would form an inversion, and thus a $\boldsymbol{B}$-block. This defines the floors. Also, this permits to bound the length of outputs outside these floors: a large output outside floors would imply an inversion.

This terminates the proof that **P2** $\to$ **P3** for sweeping transducers. We will prove **P3** $\to$ **P1** for the general case in the next section.

3.4.3 General case

Let us now prove Theorem 3.3, i.e. **P1** $\to$ **P2** $\to$ **P3** $\to$ **P1**, for arbitrary functional two-way transducers (not necessarily sweeping). We fix such a transducer $\mathcal{T}$ for this section, and a successful normalized run ρ of it on an input word u. We also keep the constant $\boldsymbol{H} = 2|Q| - 1$ as the length of the maximal crossing sequence in ρ. The proof follows the same lines as for sweeping transducers, but with additional difficulties concerning loops. In order to retrieve word equations similar to the sweeping case, we define the notions of *idempotent loops* and their *components*.

Idempotent loops and components. Let us analyze slices of runs defined by intervals of the input word (not necessarily loops), and the way they can be composed. Such a slice on an interval I can be abstracted by three elements: the two crossing sequences c_1, c_2 at the borders, and the internal shape, i.e. how locations at the borders are linked. We name this shape the flow F_I of the interval I, and these three elements its *effect* $E_I = (F_I, c_1, c_2)$.

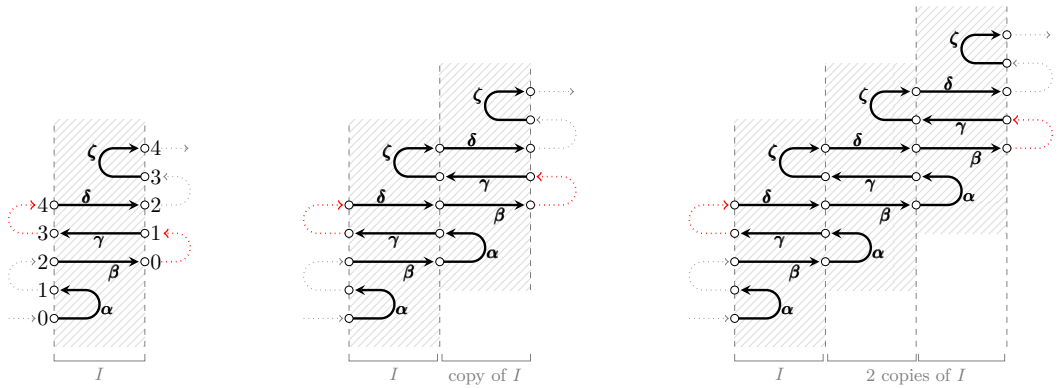


Figure 3.10: Pumping a loop in a two-way run.

Formally, the *flow* F_I of the interval $I = [x_1, x_2]$ of positions of the input word is the graph which nodes are $\{0, \dots, h\}$, where h is the maximal length of the two crossing sequences at x_1 and x_2, and where an edge $y \to y'$ denotes that there is a factor of the run ρ that starts at (x, y) and ends at (x', y'), where x and x' are at the border of I ($\{x, x'\} \subseteq \{x_1, x_2\}$). Consider for instance the interval I on the left of Figure 3.10. The flow F_I has nodes $\{0, \dots, 4\}$ and edges $0 \to 1 \to 3 \to 4 \to 2 \to 0$.

Then we define finite semigroups for flows and effects. Indeed, from two flows F and F', one can define the new flow $F \circ F'$ obtained by plugging F and F' if possible, or a special element $\perp$ if not. We do not formalize this operation here, but the definition follows the intuition. For instance the flow $F_I \circ F_I$ is illustrated in the middle of Figure 3.10. From two effects $E = (F, c_1, c_2)$ and $E' = (F', c'_1, c'_2)$, one defines $E \odot E' = (F \circ F', c_1, c'_2)$ whenever $c_1 = c'_2$ and $F \circ F' \neq \perp$, and $\perp$ otherwise.

We can now use these operations to identify loops of special interest: a loop L is *idempotent* if $E_L = E_L \odot E_L$ and $E_L \neq \perp$. For instance if we consider I as a loop in Figure 3.10 (it is the case if it has the same crossing sequences at its borders), we see that I is not idempotent. For instance $2 \to 0$ in E_I while $2 \to 3$ in $E_I \odot E_I$ (in the middle of the figure). One can check that $E_I \odot E_I$ is itself idempotent. Recall that our goal is to get word equations similar to the sweeping case when pumping loops. On non-idempotent loops like I in Figure 3.10, some factors are inverted when pumping, like the red dotted factors. We need one more notion to retrieve nice word equations: components.

A *component* of a loop L is a strongly connected component in F_L. The main property of components is that they will "group" factors that will be pumped. We can observe this in Figure 3.11, where each component is identified with a colour. A tight analysis of components shows that they form intervals in flows (they do not "interleave"), and that a left-to-right component starts with k left-to-left edges, then one left-to-right edge, and finally k right-to-right edges (and symmetrically for right-to-left components). Hence we are allowed to talk about *the crossing factor* of a component (the unique factor crossing the interval).

Inversions: P1 → P2. With idempotent loops and components, we can now obtain word equations similar to the sweeping case. For this we associate with each component the factor of ρ that will be repeated when pumping. Informally, consider the crossing factor $i \to j$ of the component C (for instance $2 \to 0$ in the blue component in Figure 3.11). Start from this edge, and follow the cycle in C ($2 \to 0 \to 1 \to 2$). Build the corresponding run factor (here it produces $\alpha_2\alpha_1\alpha_3$). This will be the pumped factor. Let us name it ρ_C.

Proposition 3.8. *Let L be an idempotent loop of ρ with components $C_1, \dots, C_k$, (listed in*

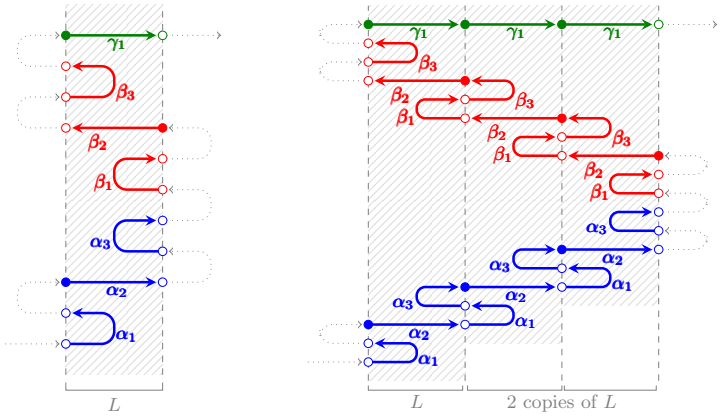


Figure 3.11: Pumping an idempotent loop with three components.

increasing order of their domains). Let ℓ_i denote the first location of the crossing factor of C_i in ρ, for $1 \leq i \leq k$. For all $n \in \mathbb{N}$, the run obtained after pumping L n times in ρ is:

$$\rho_0 \ {\rho_{C_1}}^n \ \rho_1 \ \cdots \ \rho_{k-1} \ {\rho_{C_k}}^n \ \rho_k$$

where

- *ρ_0 is the prefix of ρ that ends at ℓ_1,*
- *ρ_k is the suffix of ρ that starts at ℓ_k,*
- *ρ_i is the factor $\rho[\ell_i, \ell_{i+1}]$, for all $1 \leq i < k$.*

This can be observed in Figure 3.11, where the idempotent loop L has been pumped two times. The locations ℓ_1, ℓ_2 and ℓ_3 of the three components are indicated by filled dots. Hence $\mathsf{out}(\rho_{C_1}) = \alpha_2\alpha_1\alpha_3$. And indeed, in the pumped run, one can observe that the factor $\alpha_2\alpha_1\alpha_3$ is repeated, starting from ℓ_1. Similarly, $\mathsf{out}(\rho_{C_2}) = \beta_2\beta_1\beta_3$ and $\mathsf{out}(\rho_{C_3}) = \gamma_1$.

Let us fix the constants for the general (non-sweeping) case. We take $\boldsymbol{B} = \boldsymbol{C} \cdot \boldsymbol{H} \cdot (2^{3\boldsymbol{E}} + 4) + 4\boldsymbol{C}$, where $\boldsymbol{C}$ is the maximal size of outputs in transition rules of $\mathcal{T}$, $\boldsymbol{H} = 2|Q| - 1$ is the maximal length of crossing sequences, and $\boldsymbol{E} = (2|Q|)^{2\boldsymbol{H}}$ is the size of the effects semigroup of $\mathcal{T}$. Hence $\boldsymbol{B}$ is doubly exponential in $|\mathcal{T}|$.

Using Ramsey-type arguments on the effects semigroup (in fact, Simon's factorization forest theorem [Sim90, Col07]) we can prove that any interval I of input positions on which the output of ρ exceeds $\boldsymbol{B}$ contains an idempotent loop with non-empty output.

Let us now define inversions in the two-way case. It is very similar to the sweeping case (Definition 3.3), but we do not include the last condition on the minimality of loops, as it will better be included in the theorems.

Definition 3.4 (Inversion (two-way case))**.** *An inversion of the run ρ is a tuple $(L_1, \ell_1, L_2, \ell_2)$ where:*

1. *L_1 and L_2 are idempotent loops,*
2. *ℓ_i is the first location of the crossing factor of a component C_i of L_i (for both $i \in \{1, 2\}$),*
3. *$\ell_1 = (x_1, y_1)$ precedes $\ell_2 = (x_2, y_2)$ in ρ, while $x_1 > x_2$,*
4. *the output of ρ for C_i is non-empty (for both $i \in \{1, 2\}$).*

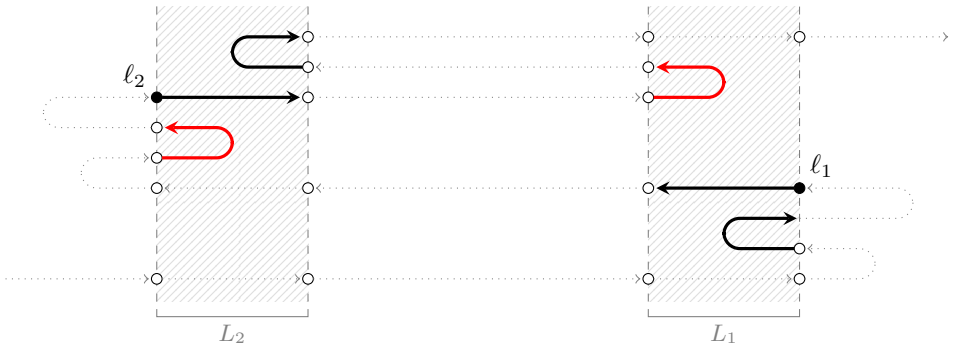


Figure 3.12: An example of an inversion $(L_1, \ell_1, L_2, \ell_2)$ of a two-way run.

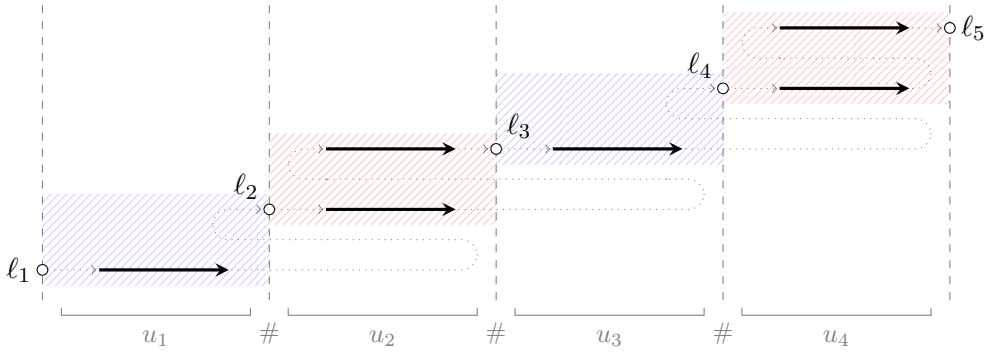


Figure 3.13: A decomposition of a run of a two-way transducer.

The definition permits situations that were not encountered in the sweeping case. Consider for instance the situation in Figure 3.12, where the non-empty outputs are indicated by red arrows. These producing factors are not "inverted", but will be when pumping. Still, this constitutes an inversion because in the definition, the inversion must occur on the crossing factors of the components, while the producing factor may be anywhere in the components (not necessarily inverted).

With this definition of inversions, we aim now at proving **P1** $\to$ **P2** with the following property **P2**, similar to the sweeping case.[4]

Proposition 3.9. *If $\mathcal{T}$ is one-way definable, then the following property* **P2** *holds:*

For all inversions $(L_1, \ell_1, L_2, \ell_2)$ of ρ, the period of $\mathsf{out}(\rho[\ell_1, \ell_2])$ is bounded by $\boldsymbol{B}$.

Like in the sweeping case (and the simple case in Section 3.2.1), the proof relies on Lemma 3.1 to bound the period of the output in the pumped run, and Lemma 3.2 to lift it to the original run. Here, minimal loops must be defined with extra care (to replace the last condition of inversion in the sweeping case), as more complex situations may arise.

Run decomposition: P2 $\to$ P3. Let us now show that **P2**, as expressed in Proposition 3.9, implies **P3**, i.e. a decomposition of the run. The proof of that part is very similar to the sweeping case, except the definition of the decomposition (diagonals and blocks), and some adaptations of the proof, that we survey here.

A *$\boldsymbol{B}$-decomposition* of a run ρ of a two-way transducer (see Figure 3.13) is still a partition of ρ into $\boldsymbol{B}$-blocks and $\boldsymbol{B}$-diagonals, where:

[4] We propose here a slightly weaker (and simpler) statement than in the complete proof, for sake of clarity.

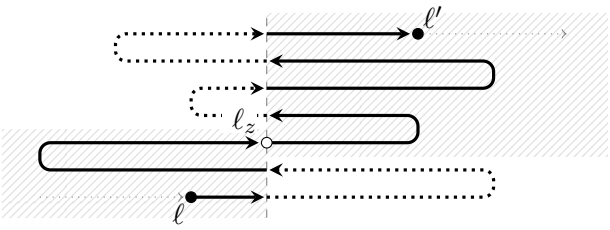


Figure 3.14: A diagonal.

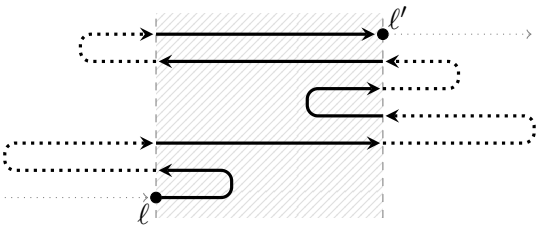


Figure 3.15: A block.

- a $\boldsymbol{B}$-diagonal (as depicted in Figure 3.14) is a factor of ρ delimited by locations $\ell = (x, y)$ and $\ell' = (x', y')$, such that for every position x'' in $[x, x']$, one can find a location $\ell_{x''}$ at that position such that the output of $\rho[\ell, \ell']$ before $\ell_{x''}$ with positions after x'' is bounded by $\boldsymbol{B}$, and also the output of $\rho[\ell, \ell']$ after $\ell_{x''}$ with positions before x'' (dotted lines in Figure 3.14).

- a $\boldsymbol{B}$-block (see Figure 3.15) is a factor of ρ delimited by locations $\ell = (x, y)$ and $\ell' = (x', y')$ which output has a period bounded by $\boldsymbol{B}$,[5] and such that the parts of this factor to the left (resp. right) of $[x, x']$ have an output bounded by $\boldsymbol{B}$ (dotted lines in Figure 3.15).

$\boldsymbol{B}$-blocks and $\boldsymbol{B}$-diagonals have a different formulation than in the sweeping case, but in fact capture similar factors. In particular, on a "sweeping run" the definitions coincide.

Now, assume that **P2** holds (every inversion has an output with bounded period, Proposition 3.9), and let us build a $\boldsymbol{B}$-decomposition (**P3**). As for the sweeping case, we define the binary relation S between locations involved in an inversion, and its reflexive transitive closure S^*. Proving that the output in a S^*-class is bounded is exactly like in the sweeping case. In order to show that such a class defines a $\boldsymbol{B}$-block, one must prove that outputs to the left (resp. right) of the block are bounded, and this requires once again Simon's factorization forest theorem [Sim90, Col07]. Proving that these $\boldsymbol{B}$-blocks do not overlap vertically is done as in the sweeping case. And finally, proving that we have $\boldsymbol{B}$-diagonals outside $\boldsymbol{B}$-blocks uses a similar line, once more with Ramsey-like arguments to show that one can find an idempotent loop when outputs are larger than $\boldsymbol{B}$. This concludes the proof that **P2** $\to$ **P3**.

Building the one-way transducer: P3 $\to$ P1. We complete the proof of Theorem 3.3 by proving that **P3** $\to$ **P1**. Let $\mathcal{T}$ be a functional two-way transducer, and D be the language of words $u \in \mathrm{dom}(\mathcal{T})$ such that all successful runs of $\mathcal{T}$ on u admit a $\boldsymbol{B}$-decomposition. We have proved (through **P2**) that if $\mathcal{T}$ is one-way definable (**P1**), then $D = \mathrm{dom}(\mathcal{T})$ (**P3**).

We describe now a procedure to build a one-way transducer $\mathcal{T}'$ from $\mathcal{T}$. This transducer is a kind of "one-way best effort", that will guess-and-check $\boldsymbol{B}$-decompositions. In that sense:

1. it will always be correct w.r.t. $\mathcal{T}$, i.e. $[\![\mathcal{T}']\!] \subseteq [\![\mathcal{T}]\!]$, and

2. its domain $\mathrm{dom}(\mathcal{T}')$ is the set of words having some successful run of $\mathcal{T}$ having a $\boldsymbol{B}$-decomposition, thus $D \subseteq \mathrm{dom}(\mathcal{T}')$.

This will prove that **P3** $\to$ **P1**: if **P3** holds then $D = \mathrm{dom}(\mathcal{T})$, and thus $[\![\mathcal{T}']\!] = [\![\mathcal{T}]\!]$, and $\mathcal{T}$ is one-way definable (**P1**).

We roughly describe how the one-way transducer $\mathcal{T}'$ can be built from the two-way transducer $\mathcal{T}$. While reading the input word, $\mathcal{T}'$ guesses a $\boldsymbol{B}$-decomposition (as in Figure 3.13), meaning that it switches between two modes: a "diagonal mode" and a "block" mode.

[5] as in the sweeping case, we allow a bounded prefix and suffix around this factor of bounded period

Before describing these modes, imagine that $\mathcal{T}'$ updates a set of pieces of two-way runs that are dynamically and non-deterministically guessed and checked, of output size bounded by $\boldsymbol{B}$, and $\mathcal{T}'$ has at most $\boldsymbol{H} = 2|Q| - 1$ of such pieces. They will be used in some places to (try to) complete a run. Let us name this the "bag" of $\mathcal{T}'$.

Assume that $\mathcal{T}'$ enters in "diagonal mode" at location ℓ (illustrated in Figure 3.14). The definition of a diagonal allows to build "stairs" (similar to the diagonals in the sweeping case): parts of the two-way run ("floors") may produce unbounded outputs, but they appear one after the other in the input order, and parts of the run in-between have an output bounded by $\boldsymbol{B}$. Hence it suffices for $\mathcal{T}'$ to (progressively) guess the positions of the floors, and guess and check the bounded output in-between using its bag, and emit those outputs when moving from one floor to the next.

Assume now that $\mathcal{T}'$ enters a "block" mode, as depicted in Figure 3.15. It has to guess the period (bounded by $\boldsymbol{B}$) of the output, and produce it progressively, by respecting both the periodicity and the size of the output, as we have already seen for instance in Example 3.1. The "bag" must be used when the two-way run uses input positions outside the block (to the left or to the right). In that case both periods must match: the period coming from the bag, and the period guessed for the block. The output can then be emitted for that part (from the definition of blocks, it is bounded by $\boldsymbol{B}$, as it is placed to the left and to the right of blocks).

Hence, every successful run of $\mathcal{T}'$ corresponds to a successful run of $\mathcal{T}$ with a $\boldsymbol{B}$-decomposition, with the same output. This shows that **P3** $\to$ **P1**.

Decidability and complexity. Now that we have proved Theorem 3.3, let us prove the main result of this section, i.e. Theorem 3.2. We have seen how to build a one-way transducer $\mathcal{T}'$ with the same properties as in the theorem. It remains to analyze the complexity.

When proving **P3** $\to$ **P1**, we defined the language D of all words inside $\mathrm{dom}(\mathcal{T})$ for which all successful runs of $\mathcal{T}$ have a $\boldsymbol{B}$-decomposition. We have seen that $\mathcal{T}$ is one-way definable iff $\mathrm{dom}(\mathcal{T}) \subseteq D$, i.e. iff $\mathrm{dom}(\mathcal{T}) \cap D^C = \emptyset$, where D^C denotes the complement of D. We will use this criterion for deciding one-way definability.

We can in fact build in 2ExpSpace (in ExpSpace if $\mathcal{T}$ is sweeping) a non-deterministic one-way automaton $\mathcal{A}$ accepting D^C, i.e. all words u for which there exists a successful run ρ on u and an inversion of ρ such that no $p \leq \boldsymbol{B}$ is a period of the output of the inversion (or u is outside $\mathrm{dom}(\mathcal{T})$). The automaton $\mathcal{A}$ has to guess the run and the inversion on-the-fly, and also guess for each $1 \leq p \leq \boldsymbol{B}$, a value $d \leq \boldsymbol{B}$ witnessing non-periodicity, i.e. the dth and the $(d+p)$th letters of the output inside the inversion differ. Hence each state of $\mathcal{A}$ requires 2ExpSpace (or ExpSpace in the sweeping case). Now, one-way definability reduces to deciding $\mathrm{dom}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$, and thus is decidable in 2ExpSpace (ExpSpace if $\mathcal{T}$ is sweeping).

Chapter 4

Resource minimization

The present chapter focuses on the following question:

> How many resources are needed to perform a transformation on a word?

This formulation is quite vague, and, to get into precise decidability problems, one needs to specify: the input (how is the transformation given?), and a definition (or measure) of the resources. This chapter is divided into two parts, depending on the input we consider: regular functions (through two-way transducers, or streaming string transducers), and then pushdown transducers.

Regular functions. In Section 4.1 we study how much resources are needed to evaluate regular functions, and more precisely transductions defined by two-way transducers or streaming string transducers (SSTs). The previous chapter was devoted to a single decidability question: does a two-way transducer admit an equivalent one-way transducer? Hence it constitutes a first answer to resource measurement, as it tells us whether a regular transduction needs to read (parts of) its input several times. However, when the two-way transducer is not one-way definable, we may want to know how it needs to process the input. In particular, we address in Section 4.1 the following questions:

1. does the two-way transducer need to reverse its head in the middle (i.e. not at the border) of the input (sweeping-definability)?
2. if not, how many times does it need to process each input position (number of sweeps)?
3. does a two-way transducer have an equivalent one with a bound on the number of reversals it performs on any word (reversal-bounded)?
4. can we build an equivalent SST with a minimum number of registers (register minimization)?

We first propose a procedure to decide if a two-way transducer is k-sweeping, when k is given as an input. Then, we show that a two-way transducer is sweeping-definable iff if is k-sweeping, for a k that depends only on the transducer, that we exhibit. Moreover, we establish a tight connection between being k-bounded reversal and being k-sweeping. This permits to answer to the first three questions. We address the last question in a restricted case, where the SST is non-deterministic, functional, but cannot concatenate the contents of two registers (named concatenation-free fNSST). This minimization relies on back-and-forth translations between concatenation-free fNSSTs and sweeping transducers.

Related work. Several direct translations between SSTs and two-way transducers have been recently proposed [DFJL17, DJR18, Led13], as already mentioned in Section 2.3.2. They can be used to relate registers of an SST and passes of a two-way transducer, but, as for now, minimization of these resources is still an open problem. The register minimization of SSTs has been addressed by Alur and Raghothaman [AR13] on a model related to deterministic SSTs (named additive cost register automata) on a unary alphabet, where registers contain integers, and updates are additions/subtractions. Daviaud, Reynier and Talbot [DRT16] propose an algorithm to compute the minimal number of registers of a deterministic SST, where updates are right-appending[1]. This model is as expressive as usual one-way transducers, but their model differs to ours in that the outputs are formed over an infinitary group. Moreover, both [AR13] and [DRT16] consider only deterministic SSTs, while we address (functional) non-deterministic ones, which may use less registers.

In this work we do not focus on the number of states of transducers, as we adopt a more "online" view. The state space minimization is an orthogonal problem, and already difficult on two-way automata (minicomplexity), as already exposed in Chapter 2.

Pushdown transducers In Section 4.2, we consider resource requirements for pushdown transducers. Our main motivation here, is to transform XML documents. Visibly pushdown transducers (VPT) are an adequate model for specifying such transformations. They operate on nested words, i.e. words where each letter is either a *call* (on which a VPT can only push) or a *return* (on which a VPT can only pop)[2]. This mimics opening and closing XML tags. A VPT reads such a word, uses its stack according to the types of letters, and outputs letters while firing transitions, like one-way transducers. XML transformations languages, like XSLT [Cla99] or XQuery [BCF+07], are usually functional and non-deterministic (when translated to transducers). For this reason we only consider *functional non-deterministic* VPTs.

In terms of resources, we focus on the amount of memory needed to perform the transduction defined by a VPT. Some simple transformations, like swapping the first and the last letters, require to store the whole input: these are typical transformations that we would like to avoid (or at least, detect). We identify three classes of transductions defined by VPTs, for which we can restrict the memory usage:

1. the first class is BM, the class of transductions that can be evaluated with *bounded memory*, i.e. memory that do not depend on the input word. We show that being in BM is decidable in CoNPTime for VPTs, in PTime for (non-pushdown) one-way transducers (it is the same as having an equivalent sequential transducer), and undecidable for (non-visibly) pushdown transducers. Bounded memory is very restrictive in this context, as it does not even allow to check that documents are well-nested [SS07].

2. the second class is HBM, which stands for *height-bounded memory*. It consists in transductions defined by VPTs, for which there exists an evaluation algorithm using an amount of space bounded by the height (and not the length) of the input nested word. The height of the nested word is, roughly speaking, its nesting depth when call/return letters are seen as brackets, or equivalently, the depth of the corresponding tree. For the class HBM, we provide a property named *height twinning property* (HTP) that captures all VPTs in HBM, and show that HTP is decidable in CoNPTime. Transductions in HBM can be evaluated with space exponential in the height of the input word.

[1] i.e. of the form $r_1 \leftarrow r_2 \cdot u$, where r_1 and r_2 are registers, and u a word.

[2] for simplicity, we discard here *internal* letters, that correspond to letters not interacting with the stack, similar to XML empty-element tags.

3. the third class is OBM, for *online bounded memory*. This corresponds to transductions defined by VPTs for which there exists an algorithm which space usage only depends, at each time point, on the current height of the input. The current height is, informally, the number of active brackets (i.e., open and non-matched) when call/return letters are seen as brackets. Like for HBM, we provide a property named *matched twinning property* (MTP) which captures exactly all VPTs in OBM, and show that it is decidable in CoNPTime. We also show that the amount of memory (for VPTs in OBM) can be limited to be quadratic in the current height. We will also see that all deterministic VPTs are in OBM, but the converse is false, in the sense that there exist VPTs in OBM that have no equivalent deterministic VPT.

Related work. VPTs are the "transducer" extension of *visibly pushdown automata* (VPAs, [AM09, Alu], also called input-driven pushdown automata [Meh80], or nested-word automata). Given a VPA, one may ask, in view of the central questions of this chapter, whether the stack is needed, i.e. whether it recognizes a regular language. This is known to be decidable in PTime [Srb09, LS19], already for the larger class of deterministic pushdown automata (all VPAs can be determinized) [Ste67, Val75]. In terms of minimization, deterministic VPAs cannot be minimized in a canonical way [AKMV05], and minimization is NP-complete [GMR20].

VPTs enjoy many desirable properties [FRR+18]. In particular functionality, and the equivalence of functional VPTs are both decidable, and VPTs are closed under regular look-ahead. The link with logics is established when allowing VPTs to operate in a two-way manner: *two-way* VPTs (with the single-use restriction) are exactly as expressive as MSOTs from nested words to words [DFRT16]. This class is also captured by *streaming tree transducers*, defined as streaming string transducers operating on a nested word, and equipped with a stack (and registers, as for SSTs) [AD17].

In terms of static analysis, VPTs can express Core XPath filters, and the height of the input word has been proved to be a lower bound for these filters [GKS07], and thus applies for VPTs. Other results and algorithms have been proposed for different settings, for instance allowing quantitative models [AKL10, AMS17].

4.1 Resources for regular functions

4.1.1 k-sweeping definability

Let us consider the k-sweeping definability problem, that is: given a functional two-way transducer, is there an equivalent k-sweeping transducer?

This problem is an extension of the one-way definability problem, which corresponds to the case $k = 1$. We present a procedure for deciding k-sweeping definability for any k, that extends the Shepherdson approach for one-way definability described in Section 3.4. In that approach, we have seen that the core concepts for one-way definability are inversions and block decompositions. Recall that, roughly, a transducer is one-way definable iff every run has a block decomposition, in which every inversion has a bounded period.

Let us fix a functional two-way transducer $\mathcal{T}$, and one of its runs ρ on an input word u. Note that our definition of two-way transducers requires that a successful run ends at the right border $\dashv$ of the input word. In this chapter we allow runs also to end at the left border $\vdash$: for instance a 2-sweeping run starts and ends at the left of the word. This allows to talk about k-sweeping transducers, even for even k's. In fact, more generally, the results and proofs in this section would also hold symmetrically for transducers starting at the right of the input word.

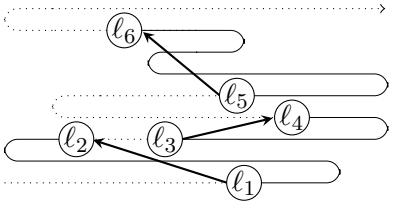


Figure 4.1: A 3-inversion.

Outline. The outline of the construction is to generalize inversions to k-inversions, such that a transducer is k-sweeping definable iff all its k-inversions are "safe". Intuitively, a k-inversion is a series of k successive inversions, that are alternatively left-to-right and right-to-left. Such a k-inversion is "safe" if (at least) one of these inversions has an output with bounded period, which will permit to do it one-way. If all k-inversions are safe, then globally each k-inversion will require at most k sweeps.

Generalized inversions. Before defining k-inversions, we define co-inversions, i.e. inversions from right to left, hence very similar to usual left-to-right inversions. Compared with Definition 3.4, only the order between x_1 and x_2 changes.

Definition 4.1 (Co-inversion)**.** *A co-inversion of the run ρ is a tuple $(L_1, \ell_1, L_2, \ell_2)$ where:*

1. *L_1 and L_2 are idempotent loops,*
2. *ℓ_i is the first location of the crossing factor of a component C_i of L_i (for both $i \in \{1, 2\}$),*
3. *$\ell_1 = (x_1, y_1)$ precedes $\ell_2 = (x_2, y_2)$ in ρ, while $x_1 < x_2$,*
4. *the output of ρ for C_i is non-empty (for both $i \in \{1, 2\}$).*

We define k-inversions as a series of k inversions / co-inversions. On Figure 4.1, a 3-inversion is depicted, through its locations ℓ_i.

Definition 4.2 (k-inversion)**.** *A k-inversion of the run ρ of the two-way transducer $\mathcal{T}$ is a sequence $\bar{\ell} = (L_1, \ell_1, L_2, \ell_2), \ldots, (L_{2k-1}, \ell_{2k-1}, L_{2k}, \ell_{2k})$ such that:*

- *ℓ_i strictly precedes ℓ_{i+1} in ρ, for all $1 \leq i \leq 2k-1$,*
- *for all even i such that $0 \leq i < k$, $(L_{2i+1}, \ell_{2i+1}, L_{2i+2}, \ell_{2i+2})$ is an inversion of ρ,*
- *for all odd i such that $0 \leq i < k$, $(L_{2i+1}, \ell_{2i+1}, L_{2i+2}, \ell_{2i+2})$ is a co-inversion of ρ.*

This generalizes the definition of inversion (Definition 3.4), as an inversion is exactly a 1-inversion. In the same way, we generalize the fact that "an inversion has a bounded period" to "a k-inversion has an inversion (or co-inversion) with a bounded period". Formally, we will say that a k-inversion $\bar{\ell}$ is ***B*-safe** if the output of ρ between ℓ_{2i+1} and ℓ_{2i+2} is bounded by $\boldsymbol{B}$ for *some* $i \in \{0, \ldots, k-1\}$.

We denote by $L_{\mathcal{T}}^{(k)}$ the language of words $u \in \mathrm{dom}(\mathcal{T})$ such that all k-inversions of all successful runs of $\mathcal{T}$ on u are ***B*-safe**. In turn, this generalizes the language D introduced to prove **P3**→**P1** in Section 3.4.3, in the sense that $D = L_{\mathcal{T}}^{(1)}$.

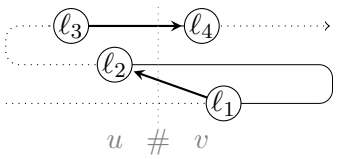


Figure 4.2: A safe 2-inversion.

Characterization. We can now provide a characterization of k-sweeping definable transductions.

Theorem 4.1. *A functional two-way transducer $\mathcal{T}$ is k-sweeping definable iff $L_{\mathcal{T}}^{(k)} = \mathrm{dom}(\mathcal{T})$, and this can be decided in* 2ExpSpace. *Moreover, one can construct in* 3ExpTime *an unambiguous k-sweeping transducer $\mathcal{T}'$ equivalent to $\mathcal{T}_{|L_{\mathcal{T}}^{(k)}}$.*

If the given transducer is already sweeping, the decision procedure is in ExpSpace *and the construction is in* 2ExpTime.

Example 4.1. *For instance, consider the function on $\Sigma = \{a, b, \#\}$ that maps every input word of the form $u\#v$ (with $u, v \in \{a, b\}^*$) to $(ab)^{|uvv|}(ba)^{|uuv|}$. It will be our running example for k-sweeping definability. This transduction can be implemented by a 3-sweeping transducer, that outputs "ab" for every letter of the first pass, and of the second pass on v, and outputs "ba" for every letter of the second pass on u, and of the third pass. This transduction can also be realized in 2 passes, where on the first pass, the transducer outputs "ab" on every letter of u, and "abab" on every letter of v, and then, on the second pass, outputs "ba" on every letter of v, and then "baba" on every letter of u. By Theorem 4.1, every 2-inversion is safe. For example, the 2-inversion of Figure 4.2 is safe, as the output of $\rho[\ell_3, \ell_4]$ has a bounded period.*

Note that in the case of an initial sweeping transducer, the 2ExpTime upper bound is tight, as we have proved in Proposition 3.2 that there exists families of sweeping-definable transductions requiring a double exponential blowup to get an equivalent one-way transducer. In this case we also obtain a first minimization algorithm:

Corollary 4.1. *When a transduction is given by a sweeping transducer, one can compute in* ExpSpace *the minimum number of passes needed by any sweeping transducer implementing it.*

The end of this section is a sketch of the proof of Theorem 4.1. We first generalize the notion of run decomposition, and relate this to $\boldsymbol{B}$-safe k-inversions. Then we show how to build the k-sweeping transducer $\mathcal{T}'$ that simulates all possible $\boldsymbol{B}$-decompositions. Finally, we show that being k-sweeping definable implies $L_{\mathcal{T}}^{(k)} = \mathrm{dom}(\mathcal{T})$.

Run decomposition. Recall that in the proof of one-way definability (Theorem 3.3), having a $\boldsymbol{B}$-decomposition, for a successful run, (**P3**) is equivalent to having a bound on the period of the output of every inversion (**P2**).

A *k-$\boldsymbol{B}$-decomposition* of a successful run ρ of a two-way transducer $\mathcal{T}$ is a sequence of locations $\bar{\ell} = \ell_0, \ell_1, \ldots, \ell_k$ of ρ such that:

- ℓ_0 (resp. ℓ_k) is the first (resp. last) location of ρ, and ℓ_i strictly precedes ℓ_{i+1} in ρ, for all $0 \leq i < k$,
- for all even indexes i, with $0 \leq i < k$, and all inversions (L, ℓ, L', ℓ') of ρ, with ℓ, ℓ' between ℓ_i and ℓ_{i+1} in ρ, the output of $\rho[\ell, \ell']$ has period at most $\boldsymbol{B}$,
- for all odd indexes i, with $1 \leq i < k$, and all co-inversions (L, ℓ, L', ℓ') of ρ, with ℓ, ℓ' between ℓ_i and ℓ_{i+1} in ρ, the output of $\rho[\ell, \ell']$ has period at most $\boldsymbol{B}$.

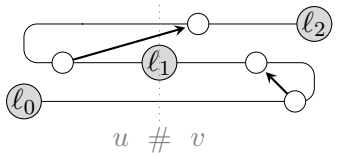


Figure 4.3: A 2-$\boldsymbol{B}$-decomposition.

In other words, a k-$\boldsymbol{B}$-decomposition is a partition of ρ into k parts, that are alternatively containing inversions with periodicity of the output bounded by $\boldsymbol{B}$, and co-inversions with the same property.

Example 4.2. *Let us illustrate this definition on our running example for k-sweeping definability (Example 4.1). A typical 2-$\boldsymbol{B}$-decomposition is ℓ_0, ℓ_1, ℓ_2 as depicted in Figure 4.3. Intuitively, it means that $\rho[\ell_0, \ell_1]$ can be done in one left-to-right pass, and $\rho[\ell_1, \ell_2]$ in a right-to-left pass.*

In fact, it almost follows from the definitions that, for every $u \in \mathrm{dom}(\mathcal{T})$, $u \in L^{(k)}_{\mathcal{T}}$ iff every successful run of $\mathcal{T}$ on u has a k-$\boldsymbol{B}$-decomposition. In order to obtain a decision procedure, we encode k-decompositions as words, by annotating the input. Hence, given a run ρ of $\mathcal{T}$ on u, and a sequence $\bar{\ell} = \ell_0, \ell_1, \ldots, \ell_m$ of locations of ρ, we define $\langle u, \rho, \bar{\ell}\rangle$, obtained from u by annotating it with the crossing sequence at each position, and also with the m-tuple $\bar{y} = (y_1(x), \ldots, y_m(x))$, where $y_i(x)$ is y' if $\ell_i = (x, y')$, or $\perp$ if ℓ_i does not appear at position x. In the sequel, m will always be bounded by k, and every $y_i(x)$ is also bounded by the crossing degree of $\mathcal{T}$.

We show that the language $F^{(k)}_{\mathcal{T}}$ of all words $\langle u, \rho, \bar{\ell}\rangle$ corresponding to k-$\boldsymbol{B}$-decompositions of runs of $\mathcal{T}$ can be recognized by a one-way automaton of size triply exponential in $|\mathcal{T}|$, that can be built on-the-fly in double exponential space. The same holds for the complement $\overline{F^{(k)}_{\mathcal{T}}}$ of $F^{(k)}_{\mathcal{T}}$. The proof reduces the problem to one-way definability (and thus Theorem 3.2), by studying the transducers $\mathcal{T}_i$ that behave like $\mathcal{T}$ between ℓ_i and ℓ_{i+1}, and output nothing outside this interval. These transducers must be one-way definable, alternatively left-to-right and right-to-left.

We can now decide in 2ExpSpace whether $L^{(k)}_{\mathcal{T}} = \mathrm{dom}(\mathcal{T})$, as stated in Theorem 4.1: We always have $L^{(k)}_{\mathcal{T}} \subseteq \mathrm{dom}(\mathcal{T})$, so it remains to test whether $L^{(k)}_{\mathcal{T}} \supseteq \mathrm{dom}(\mathcal{T})$. We have seen that $L^{(k)}_{\mathcal{T}}$ coincides with the words having a k-$\boldsymbol{B}$-decomposition, and thus with the projection of $F^{(k)}_{\mathcal{T}}$ on its first component. Thus $L^{(k)}_{\mathcal{T}} \supseteq \mathrm{dom}(\mathcal{T})$ iff $\overline{F^{(k)}_{\mathcal{T}}} \cap \{\langle u, \rho, \bar{\ell}\rangle \mid u \in \mathrm{dom}(\mathcal{T})\} = \emptyset$ which can be checked in double exponential space by building the automaton recognizing $\overline{F^{(k)}_{\mathcal{T}}}$ on-the-fly.

Building $\mathcal{T}'$. Let us show the right-to-left implication of Theorem 4.1 by building a k-sweeping transducer $\mathcal{T}'$ equivalent to $\mathcal{T}$, assuming that $L^{(k)}_{\mathcal{T}} = \mathrm{dom}(\mathcal{T})$. The main idea is to guess a k-$\boldsymbol{B}$-decomposition, and between each of the corresponding location, build the one-way transducer obtained in Theorem 3.2 to perform one pass, and concatenate all these "one-way transducers" to build the k-sweeping transducer $\mathcal{T}'$.

The main difficulty here is to be able to consider all these parts of the run independently: The naive approach would plug parts of different runs that could not form a real run together. To circumvent this, we define a lexicographical order on runs ρ, and a notion of maximal k-$\boldsymbol{B}$-decomposition $\bar{\ell}$. Hence we can identify a canonical run ρ and k-$\boldsymbol{B}$-decomposition $\bar{\ell}$ among all $\langle u, \rho, \bar{\ell}\rangle$ associated with a given input u. These can be filtered using a one-way automaton of size doubly exponential in $\mathcal{T}$. Hence, $\mathcal{T}'$ guesses on-the-fly a run ρ and a k-$\boldsymbol{B}$-decomposition $\bar{\ell}$, checks in parallel that these are canonical, builds a one-way transducer T_i (left-to-right or right-to-left) using Theorem 3.2 inside each of the k parts of the decomposition, and concatenates

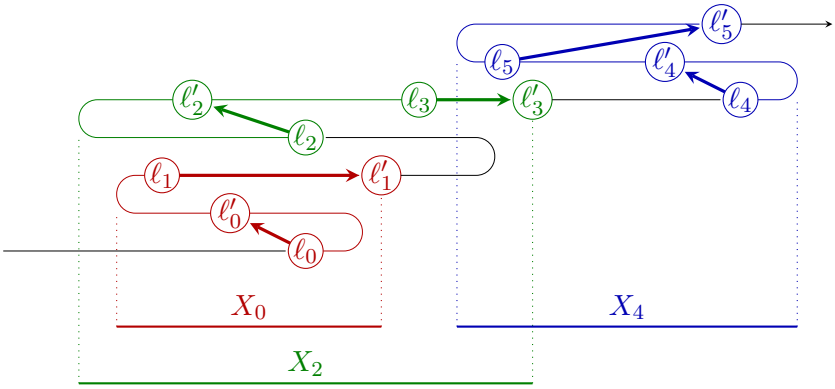


Figure 4.4: A 6-inversion, and related intervals X_i used in the proof of Proposition 4.1.

them to form $\mathcal{T}'$. The triple exponential size of $\mathcal{T}'$ comes from using Theorem 3.2 (double exponential if $\mathcal{T}$ is sweeping).

k-sweeping implies safe k-$\boldsymbol{B}$-decompositions. To complete the proof of Theorem 4.1, it remains to show that, if $\mathcal{T}$ has an equivalent k-sweeping transducer $\mathcal{T}'$, then $L^{(k)}_{\mathcal{T}} = \mathrm{dom}(\mathcal{T})$. In other terms, for every input word u, every run ρ of $\mathcal{T}$ on it, and every k-inversion $\bar{\ell} = (L_1, \ell_1, L_2, \ell_2), \ldots, (L_{2k-1}, \ell_{2k-1}, L_{2k}, \ell_{2k})$ of ρ, there is an inversion (or co-inversion) among $\bar{\ell}$ having an output with a period bounded by $\boldsymbol{B}$. Intuitively, we have to identify two locations ℓ_{2i+1}, ℓ_{2i+2} such that the output of ρ between them is entirely covered by a single pass of $\mathcal{T}'$.

This point is more technical than it seems, because there is no a priori one-to-one correspondence between the run ρ of $\mathcal{T}$ and a run of $\mathcal{T}'$: they are not "synchronized", have different shapes, etc. The correspondence is established by pumping both L_{2i+1} and L_{2i+2}, and also some loops in $\mathcal{T}'$ that yield the same output. In fact we need to pump all loops of $\bar{\ell}$ and find correspondences with loops of $\mathcal{T}'$. Then we use combinatorial tools to show that $(L_{2i+1}, \ell_{2i+1}, L_{2i+2}, \ell_{2i+2})$ is $\boldsymbol{B}$-safe.

4.1.2 Sweeping-definability (and bounded-reversal)

In the previous section, we described a procedure to decide, for a given two-way transducer, whether there exists an equivalent k-sweeping transducer, and thus minimize the number of passes of any sweeping transducer.

Sweeping definability. Let us now focus on the same problem but when k is not given, i.e. on deciding whether a given two-way transducer is sweeping definable. Given Theorem 4.1, it suffices to exhibit an upper bound on such a k.

Proposition 4.1. *A functional two-way transducer is sweeping definable iff it is k-sweeping definable for $k = 2\boldsymbol{H}(2^{3\boldsymbol{E}} + 1)$.*

We recall that $\boldsymbol{H} = 2|Q| - 1$ is the maximal length of a crossing sequence, and that $\boldsymbol{E} = (2|Q|)^{2\boldsymbol{H}}$ is the size of the effects semigroup (see Section 3.4.3). We prove Proposition 4.1 by contradiction, assuming that $\mathcal{T}$ is not k-sweeping definable for $k = 2\boldsymbol{H}(2^{3\boldsymbol{E}}+1)$. We show that in this case, it is not m-sweeping definable, for every $m > 0$. By Theorem 4.1, $L^{(k)}_{\mathcal{T}} \neq \mathrm{dom}(\mathcal{T})$, which means, as we have seen, that there exists a run ρ and a k-inversion in ρ which is not $\boldsymbol{B}$-safe. To each pair of successive inversion/co-inversion of this k-inversion, we associate the interval of positions of the input word visited in the inversion, the co-inversion, and the run

in-between (denoted X_i in Figure 4.4). As ρ is $\boldsymbol{H}$-crossing, we can take one such interval every $\boldsymbol{H}$ (after sorting them by their maximal position), and obtain a subset $\mathcal{X}$ of such intervals that are pairwise disjoint, and such that $|\mathcal{X}| = 2^{3\boldsymbol{E}} + 1$. Now, using Simon's factorization forest theorem [Sim90, Col07], we can show that, among the leftmost positions of intervals in $\mathcal{X}$, three of them delimit consecutive idempotent loops. By pumping these loops, we can build runs of $\mathcal{T}$ with m-inversions that are, themselves, not $\boldsymbol{B}$-safe. By Theorem 4.1, this means that $\mathcal{T}$ is not m-sweeping definable, and this holds for every $m > 0$.

The decidability is a direct consequence of Theorem 4.1 and Proposition 4.1.

Theorem 4.2. *It is decidable whether a functional two-way transducer is sweeping definable.*

Bounded reversal. When considering arbitrary runs of functional two-way transducers, we can observe that the number of reversals is not bounded, even if the crossing degree is bounded. This is typically illustrated by the shape of "stairs", like in Figure 3.13 for instance.

In fact, two-way transducers having a bounded number of reversals in all of their runs corresponds exactly to sweeping definable transducers.

Theorem 4.3. *Every functional two-way transducer with at most $k-1$ reversals per run can be transformed into an equivalent k-sweeping transducer, and conversely.*

As a consequence, functions that can be implemented by a functional two-way transducers with a bounded number of reversals per run are exactly those definable by sweeping transducers.

One direction is straightforward: if a transducer is k-sweeping for some k, it is clearly $(k-1)$-reversal bounded. The other direction is more involved, and amounts to show how reversals of a transducer with $k-1$ reversals per run can be performed at the borders of the input word, instead of inner positions. For now, let us assume that this transducer $\mathcal{T}$ is unambiguous (we will discuss later how to lift this condition). The main idea is to keep the original run of $\mathcal{T}$, but, on inner reversals, keep the current direction until the next border, and then come back to the position where we left the inner reversal, and continue the original run. The difficulty here is to identify the position of the inner reversal when we come back to it. We solve this problem by maintaining, in the sweeping transducer $\mathcal{S}$, the whole crossing sequence of the original run, also pointing the position of the current run of $\mathcal{T}$ (i.e. the current level) in it. This is somehow similar to Shepherdson's construction (see Section 3.1.1) but on every pass of $\mathcal{S}$. This ensures that a simulated run, if successful, is a correct one for $\mathcal{T}$. By unambiguity of $\mathcal{T}$, it is exactly *the* successful run of $\mathcal{T}$ on the input word, and this holds for every pass of the sweeping transducer $\mathcal{S}$, which, in turn, ensures that $\mathcal{S}$ is unambiguous, and thus equivalent to $\mathcal{T}$. It remains to deal with the case where $\mathcal{T}$ is not unambiguous. In this case, we use the same construction, but keep *sets of* crossing sequences in the states of $\mathcal{S}$, in order to only consider the least run of $\mathcal{T}$ on u, and thus recover the properties of unambiguity.

Corollary 4.2. *It is decidable whether a functional two-way transducer has an equivalent one with a bounded number of reversals.*

This is a direct consequence of Theorem 4.3 and Theorem 4.2.

4.1.3 Register minimization of concatenation-free SST

Let us now consider resources of functional non-deterministic streaming string transducers (*fNSSTs*), and more precisely the number of registers required to implement a transduction. This problem is open for SSTs in general. We focus here on a particular case, where the concatenation of registers is forbidden in register updates.

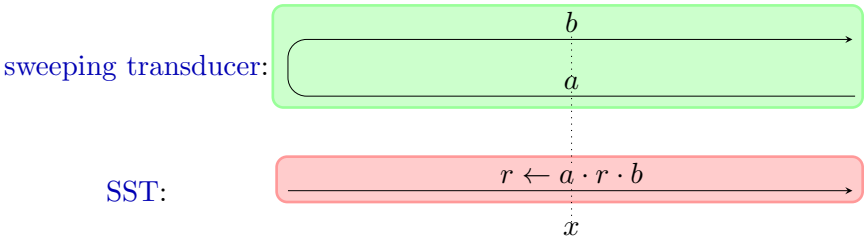


Figure 4.5: Corresponding steps in an R-sweep and in a concatenation-free SST.

Recall from Section 2.3 that an fNSST $\mathcal{T}$ has a set of registers R and that these are updated through updates $up \in \nabla$ in such a way that, for a register $r \in R$, $up(r) \in (\Delta \uplus R)^*$, where Δ is the output alphabet. An fNSST is concatenation-free if every update uses at most one register in its image, i.e. $up(r) \in \Delta^* \cdot (R \cup \{\epsilon\}) \cdot \Delta^*$, for every $r \in R$ and every $up \in \nabla$. Moreover we will consider sweeping transducers starting (and ending) at the *right* of the input word, and name them *R-sweeping transducers*.

Theorem 4.4. *Every concatenation-free* fNSST *with k registers can be translated into an equivalent $2k$-pass R-sweeping transducer, and conversely.*

The conversion of a concatenation-free fNSST *to an equivalent $2k$-pass R-sweeping transducer is in* ExpTime*, while the converse is in 2*ExpTime.

Assuming (for now) that there is no swap between registers, and that transducers are unambiguous, the key observation is the following:

> the way the output is produced in a sweep starting from the right (i.e. right-to-left then left-to-right, named R-sweep in the following) is exactly the same as updates of a register.

This is illustrated in Figure 4.5: if, at position x, the sweep outputs a on its first pass and b on the second, then an fNSST with one register r can update it with $a \cdot r \cdot b$. And this holds in the other direction: from an fNSST with one register r and such an update, we can build an R-sweeping transducer outputting a on its first pass, and b on the second at this position.

When several registers are used, its suffices to use one sweep per register. Consider for instance the function $u \mapsto u \cdot \text{mirror}(u) \cdot u$ over $\{a, b\}^*$. It requires 4 sweeps for any R-sweeping transducer, and 2 registers for any concatenation-free fNSST implementing it. We show this correspondence in Figure 4.6. The fNSST is composed by two registers r_1 and r_2, where r_1 is used to store the input u, and r_2 the word $\text{mirror}(u) \cdot u$, using the following updates, for every $\sigma \in \{a, b\}$:

$$r_1 \leftarrow r_1 \cdot \sigma \qquad\qquad r_2 \leftarrow \sigma \cdot r_2 \cdot \sigma$$

We only use one state q, and set $out(q) = r_1 r_2$. On the R-sweeping transducer, the first sweep simulates r_1, by outputting ϵ in the first pass (to the left of r_1 in the update), and σ in the second pass (to the right of r_1 in the update). The second sweep simulates r_2, following the same principle: it outputs σ in both passes (to the left and right of r_2 in the update).

Swapping registers, and unambiguity. Some details have to be considered, though. First, registers of an fNSST may swap, as for instance in the update $r_1 \leftarrow r_2 \cdot a$. Let us explain why this is not problematic when the fNSST is concatenation-free. The sweeping transducer first guesses which final state q will be reached, and hence which registers will be used in the output $out(q)$. Assume for instance that $out(q) = a \cdot r_2 \cdot b \cdot r_1$. The sweeping transducer outputs a, then performs a sweep that will output the content of r_2 in the fNSST, then b, then an additional

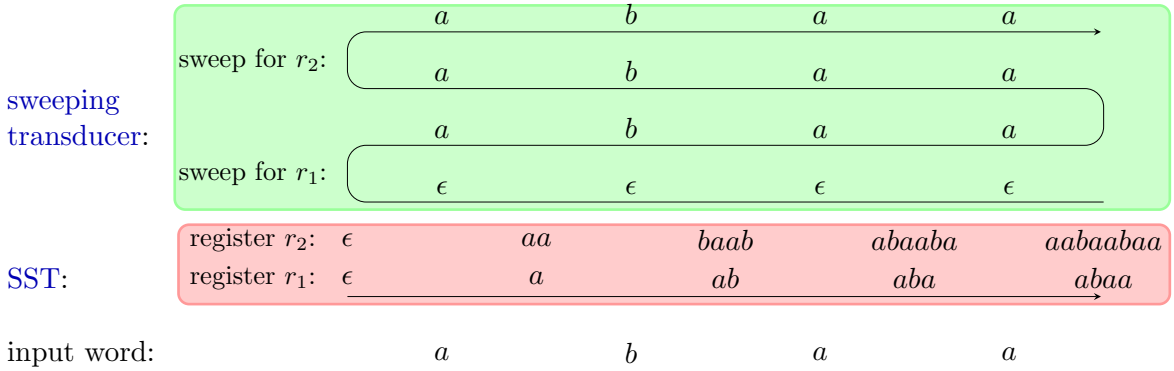


Figure 4.6: Corresponding runs of a sweeping transducer and of an SST.

sweep outputting the content of r_1 in the fNSST. The first sweep, outputting the content of r_2, may at some point have to swap to (the simulation of) r_1, if for instance the update $r_2 \leftarrow r_1$ happens. As the fNSST is both copyless and concatenation-free, this is not problematic: there is at most one register to simulate at a time.

For now we assumed that both the fNSST and the sweeping transducer are unambiguous. This is necessary in the translations (in both directions) presented above. In particular the sweeping transducer simulating an fNSST checks, at every *pass*, that the guessed run of the fNSST is the correct one, and then stays coherent from one pass to the other. This argument is very similar to the one presented for the bounded-reversal case (Theorem 4.3), and can be lifted in a similar way, by establishing a lexicographical order on runs. In the other direction, the fNSST simulating a sweeping transducer uses a crossing sequence construction to simulate all the sweeps in one pass, and check at the end that the run is the correct one.

In terms of complexity, the conversion of a concatenation-free fNSST is in PTime when it is unambiguous, and becomes in ExpTime when it is not. The conversion of an R-sweeping transducer is in ExpTime when it is unambiguous (due to the crossing sequence construction), and in 2ExpTime when it is not.

Finally, as consequence of Theorem 4.4 and Corollary 4.1, we obtain:

Corollary 4.3. *Given a concatenation-free* fNSST*, one can compute the minimal number of registers required by any concatenation-free* fNSST *implementing it.*

Let us conclude with the following remark: concatenation-free fNSSTs have the same expressive power as fNSSTs with a bounded number of register concatenations in all runs. Indeed, in that case, every register concatenation can be replaced by a new register.

4.2 Resources of pushdown transducers

We now move to a richer transducer model including a stack, and study related resource requirements.

4.2.1 Pushdown transducers, and streaming setting

Nested words. Up to now, we considered finite words on a finite alphabet Σ, as simple sequences of elements in Σ. A *nested alphabet* is a finite alphabet Σ partitioned into three disjoint sets Σ_c, Σ_r and Σ_ι that we call respectively *call*, *return* and *internal* alphabets. A *nested word* is a word over a nested alphabet. The set of *well-nested* words Σ^*_{wn} over the nested

alphabet Σ is the smallest set of words containing Σ_ι^*, and such that for all $c \in \Sigma_c$, all $r \in \Sigma_r$, and all $u, v \in \Sigma_{\mathsf{wn}}^*$, $curv \in \Sigma_{\mathsf{wn}}^*$.

The *current height* $\mathsf{hc}(u)$ of a prefix u of a well-nested word is the number of unmatched calls: if u is well-nested and $c \in \Sigma_c$, then $\mathsf{hc}(u) = 0$, and $\mathsf{hc}(vcu) = \mathsf{hc}(v) + 1$. The *height* $\mathsf{h}(u)$ of u is its maximal current height while reading it: $h(u) = \max_{v \preceq u} \mathsf{hc}(v)$.

Visibly pushdown automata and transducers. A *visibly pushdown automaton* (VPA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \Gamma, \delta, I, F)$ where Q, I and F are states defined as for automata, Σ is a nested alphabet, Γ is a finite set of *stack symbols*, and $\delta = \delta_c \uplus \delta_r \uplus \delta_\iota$ is the transition relation, partitioned into call, return and internal transition relations: $\delta_c \subseteq Q \times \Sigma_c \times \Gamma \times Q$, $\delta_r \subseteq Q \times \Sigma_r \times \Gamma \times Q$, and $\delta_\iota \subseteq Q \times \Sigma_\iota \times Q$. A VPA is *deterministic* if:

- for every $(q, c) \in Q \times \Sigma_c$, there exists at most one $(\gamma, q') \in \Gamma \times Q$ such that $(q, c, \gamma, q') \in \delta_c$,
- for every $(q, r, \gamma) \in Q \times \Sigma_r \times \Gamma$, there exists at most one $q' \in Q$ such that $(q, r, \gamma, q') \in \delta_r$,
- and for every $(q, \iota) \in Q \times \Sigma_\iota$, there exists at most one $q' \in Q$ such that $(q, \iota, q') \in \delta_\iota$.

A *configuration* of $\mathcal{A}$ is a pair (q, σ) where $q \in Q$ is the current state, and $\sigma \in \Gamma^*$ is the current stack content (the bottom of the stack is placed on the left, and we write $\bot$ for the empty stack). A *run* of $\mathcal{A}$ on a well-nested word $u = u_1 \cdots u_n$ is a sequence of configurations $(q_i, \sigma_i)_{0 \le i \le n}$ such that, for all $1 \le i \le n$, either:

- $(q_{i-1}, u_i, \gamma, q_i) \in \delta_c$ and $\sigma_i = \sigma_{i-1}\gamma$, or
- $(q_{i-1}, u_i, \gamma, q_i) \in \delta_r$ and $\sigma_{i-1} = \sigma_i\gamma$, or
- $(q_{i-1}, u_i, \gamma, q_i) \in \delta_\iota$ and $\sigma_i = \sigma_{i-1}$.

Given a run on u, we associate with each pair of successive configurations $(q_{i-1}, \sigma_{i-1}), (q_i, \sigma_i)$ the transition rule t_i permitting it. Such a run is successful if $q_0 \in I$, $q_n \in F$ and $\sigma_0 = \sigma_n = \epsilon$.

A *visibly pushdown transducer* (VPT) $\mathcal{T}$ is a pair $(\mathcal{A}, \mathsf{out})$ where $\mathcal{A}$ is a visibly pushdown automaton, and $\mathsf{out} : \delta \to \Delta^*$ is the output function of $\mathcal{T}$. A *run* ρ of $\mathcal{T}$ on a well-nested word $u = u_1 \cdots u_n$ is a run $(q_i, \sigma_i)_{0 \le i \le n}$ of $\mathcal{A}$ on u, and its associated output is $\mathsf{out}(t_1) \cdots \mathsf{out}(t_n)$ where t_i is the transition rule associated with $(q_{i-1}, \sigma_{i-1}), (q_i, \sigma_i)$, for every $1 \le i \le n$.

A VPT is *deterministic* if its associated VPA is. We will also require that the VPT are reduced, in the following sense. A configuration (q, σ) of a VPT $\mathcal{T}$ (i.e. of its underlying VPA) is *accessible* (resp. *VPA co-accessible*) if there exists a word u and a run of $\mathcal{T}$ on u starting in a configuration $(q_I, \bot)$ with $q_I \in I$ (resp. starting in (q, σ)) and ending in (q, σ) (resp. ending in $(q_F, \bot)$ for some $q_F \in F$). A VPT is *reduced* if every accessible configuration is also co-accessible. From a given VPT, one can always compute an equivalent reduced VPT in polynomial time [CRT15]. For the other automata-related notions, we use the same terminology as in the context of other automata and transducers, with definitions adapted in a straightforward manner. We will mostly consider *functional* VPTs in this manuscript. Functionality of VPTs can be decided in polynomial time [FRR+18], but the determinizability question is still open:

Open problem 2 (VPT *determinization*)

> *Find an algorithm that, given a functional VPT, decides whether there exists an equivalent deterministic VPT.*

For a given $k \in \mathbb{N}$ and a given VPT $\mathcal{T}$, we define FST$(\mathcal{T}, k)$ the one-way transducer equivalent to $\mathcal{T}$, but restricted to input words of height less than k. Its states are configurations (q, σ) of $\mathcal{T}$ where q is a state of $\mathcal{T}$ and $\sigma \in \Gamma^*$, with $|\sigma| \le k$.

Turing transducers. In this section we are interested in measuring the space complexity of any algorithm implementing certain kinds of functional transductions. Let us define Turing transducers, a machine model that corresponds to programs implementing transductions.

A *Turing transducer* is a Turing machine with the following architecture:

- a read-only left-to-right *input tape* over some finite alphabet Σ,
- a write only left-to-right *output tape* over some finite alphabet Δ, and
- a working tape over some finite alphabet Σ'.

The transitions of this machine are deterministic, and the space complexity is measured on the working tape only.

We say that a Turing transducer *computes* a (partial) function $f : \Sigma^* \to \Delta^*$ if, for all words $u \in \mathrm{dom}(f)$, when u is placed on the input tape, the computation halts in some accepting state and the content of the output tape is $f(u)$, while for all words $u \notin \mathrm{dom}(f)$, when u is placed on the input tape, the computation halts in some rejecting state. In that case we say that f is computable by this Turing transducer.

4.2.2 Bounded memory

We say that a partial function $f : \Sigma^* \to \Delta^*$ is *bounded memory* (BM) if there exists a constant $k \geq 0$ and a Turing transducer that computes f, and runs in space complexity at most k.

Finite state transducers. Recall that deterministic transducers are one-way transducers with a deterministic underlying automaton, and they can append a suffix to their output once the last state is reached. For finite state (one-way) transducers, deterministic transducers can clearly be evaluated with bounded memory (only the current state needs to be stored). The converse also holds:

Proposition 4.2. *Let $\mathcal{T}$ be a functional one-way transducer. Then $[\![\mathcal{T}]\!]$ is in BM iff $\mathcal{T}$ has an equivalent deterministic transducer. This is decidable in* PTime.

Indeed, if $[\![\mathcal{T}]\!]$ is in BM, we can consider the Turing transducer M with memory bounded by k computing it. As M is deterministic, we can consider it as a deterministic transducer with states of the form (q, u) where q is the state of M, and u the content of the working tape (with $|u| \leq k$). The PTime complexity comes from the decision procedure for sequentiality, based on the twinning property [WK95, BC02, BCPS03].

Pushdown transducers. The situation differs for pushdown transducers.

Proposition 4.3. *It is undecidable whether a transduction defined by a non-deterministic pushdown transducer is in BM.*

Indeed, this decision problem is harder than testing regularity of non-deterministic pushdown automata, which is known to be undecidable [BHPS61, GR63]. This is witnessed by the reduction that, given a pushdown automaton $\mathcal{A}$, associates the transduction $\mathcal{T}$ computing the identity on $\mathcal{L}(\mathcal{A})$. Then $[\![\mathcal{T}]\!]$ is in BM iff $\mathcal{L}(\mathcal{A})$ is regular (with the same kind of construction as for finite-state transducers above).

In the deterministic case, the regularity test of pushdown automata becomes decidable [Ste67, Val75], thus the previous proof does not apply. The BM membership is in fact open.

Open problem 3 (*Deterministic pushdown transducers in bounded memory*)

> *Is the following problem decidable: Given a deterministic pushdown transducer $\mathcal{T}$, is $[\![\mathcal{T}]\!]$ in BM?*

Visibly pushdown transducers. For visibly pushdown transducers, the membership to BM becomes decidable again.

Proposition 4.4. *If $\mathcal{T}$ is a functional VPT with n states, then $[\![\mathcal{T}]\!]$ is BM iff*

- *for all $u \in \mathrm{dom}(\mathcal{T})$, $\mathsf{h}(u) \leq n^2$ and*
- *$[\![\mathrm{FST}(\mathcal{T}, n^2)]\!]$ is BM.*

Moreover, this is decidable in CoNPTime.

This means that being bounded memory is very restrictive for visibly pushdown transducers: it can only accept words of bounded height (this is obtained by standard pumping arguments). A decision procedure could be obtained by building $\mathrm{FST}(\mathcal{T}, n^2)$ and then using Proposition 4.2, but this device is of exponential size. We will see in Section 4.2.4 that, for VPTs with input words of bounded height, being BM and HBM is equivalent, and HBM can be tested in CoNPTime.

Bounded memory is a very strong restriction, that makes little sense when dealing with nested words: in general they need at least an unbounded stack to be parsed. We now consider larger classes, observing how much memory is needed, compared to the stack height.

4.2.3 An online algorithm for VPT evaluation

In this section, we describe an evaluation algorithm, that takes a functional VPT $\mathcal{T}$ as input, reads an input word u letter by letter, and outputs progressively $[\![\mathcal{T}]\!](u)$. This algorithm will be used later to obtain upper bounds for the classes HBM and OBM of VPTs.

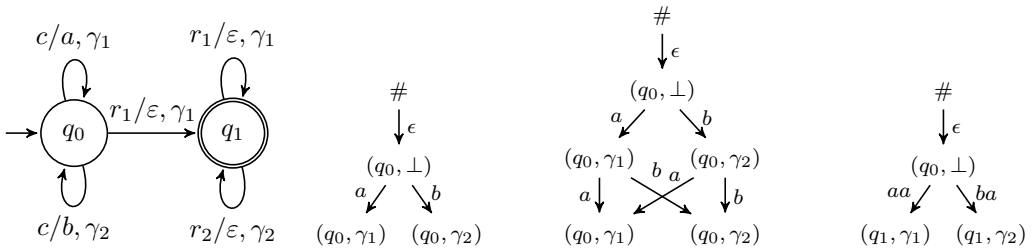
Naive version. Let us start with a naive approach. Recall that $\mathcal{T}$ is functional but non-deterministic: the algorithm has to store potential candidates for the output. Recall also that a VPT is reduced if all its accessible configurations are also co-accessible. The first step of the algorithm is to compute an equivalent reduced VPT, and this can be obtained in PTime [CRT15]. Hence from now on we assume $\mathcal{T}$ to be reduced. A consequence of being reduced and functional is that, if two runs of $\mathcal{T}$ reach a common configuration (q, σ), then they produced the same output word v so far. This means that we can just store triples (q, σ, w) where $(q, \sigma) \in Q \times \Gamma^*$ is a configuration of $\mathcal{T}$, and $w \in \Delta^*$ is (the prefix of) a candidate output word.

At the beginning, this set is $\{(q, \bot, \epsilon) \mid q \in I\}$. At each incoming letter of u, each configuration is updated (or dies) according to the transition rules of $\mathcal{T}$, appending new output letters to words w. Once the input word has been read, final configurations share the same output w, which can be yielded.

This algorithm has two main weaknesses. The first one is that the number of stored candidates may be exponential in the size of the input. The second one is that all candidate output words are stored until the end. They could be output on-the-fly, if they share a common prefix. We address these two issues in the sequel.

Storing configurations. In order to avoid the exponential blowup, we use a DAG to store the set of alive configurations in a compact manner. Let us illustrate this on an example. Consider the VPT $\mathcal{T}_1$ represented in Figure 4.7 (a). The DAG obtained after reading c (resp cc, ccr_1) is illustrated in Figure 4.7 (b) (resp. (c), (d)). Each configuration (q, σ, w) is stored along a branch of the DAG: the current state q is at the leaf, the output w is stored on the edges, and the stack content σ is obtained by concatenating stack symbols in the node of the branch. For instance, in the DAG of Figure 4.7 (c), the branch

$$\# \xrightarrow{\epsilon} (q_0, \bot) \xrightarrow{b} (q_0, \gamma_2) \xrightarrow{a} (q_0, \gamma_1)$$



(a) VPT $\mathcal{T}_1$. (b) After reading c. (c) After reading cc. (d) After reading ccr_1.

Figure 4.7: Data structure used by the online algorithm.

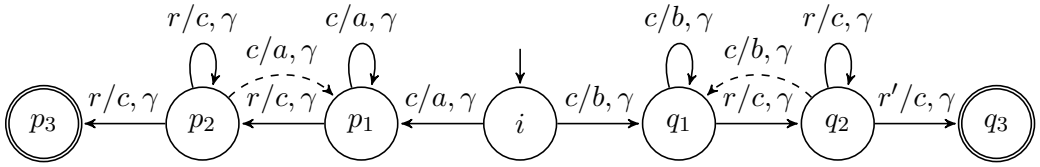


Figure 4.8: A functional VPT with $\Sigma_c = \{c\}$, $\Sigma_r = \{r, r'\}$ and $\Sigma_\iota = \{a, b\}$.

encodes the configuration $(q_0, \gamma_2\gamma_1, ba)$ of the VPT of Figure 4.7 (a).

The structure is updated in order to maintain this invariant. In particular, when a leaf cannot be updated w.r.t. an incoming input letter, it is removed from the structure, and also the orphaned nodes it created. Note that this DAG structure requires the VPT to be reduced.

Progressive output. When an input letter is read, the DAG is updated as explained above, and then a bottom-up phase computes the longest common prefix of all the outputs stored in the DAG. This prefix is output by the algorithm, and removed from the DAG. Once the prefix u' of the input u has been processed, let $\mathsf{out}_{\neq}(u')$ denote the length of the longest output among those of all configurations stored in the DAG after the update. We can measure the space complexity using this quantity.

Proposition 4.5. *Let $\mathcal{T}$ be a functional VPT. One can build in* PTIME *a Turing transducer which computes $[\![T]\!]$, and that uses space in $O((\mathsf{hc}(u')+1) \cdot \mathsf{out}_{\neq}(u'))$ on the working tape, after reading a prefix u' of a well-nested word $u \in \Sigma^*$.*

4.2.4 Height-bounded memory

As we have seen in Section 4.2.2, being bounded memory is very restrictive in the context of nested words. We relax this notion here by allowing a space usage that depends on the height of the input word (but not on its length).

We say that a (partial) function $f : \Sigma^* \to \Delta^*$ is *height-bounded memory* if there exists a function $\theta : \mathbb{N} \to \mathbb{N}$ such that f is computable by a Turing transducer that runs in space complexity at most $\theta(\mathsf{h}(u))$, on any input u.

For instance, the VPT in Figure 4.7 (a) is HBM but not BM: it stack content suffices to determine the output. Another example of functions in HBM but not in BM is the set of functions whose domain only includes (depth-first) encodings of ranked trees. If the maximal rank is k, then the length of the input word u is at most $k^{\mathsf{h}(u)}$. Let us now exhibit a function which is not HBM. This is typically the case when one can pump two candidate runs horizontally (i.e. with identical height), with different outputs, as for instance in the VPT in Figure 4.8 including dashed arrows.

Link with deterministic VPTs. Clearly, any function f implemented by a deterministic VPT is HBM. This is for instance witnessed by Proposition 4.5, where $\mathsf{out}_{\neq}(u') = 0$ for all prefixes u' when $\mathcal{T}$ is deterministic, as the output can be produced progressively (only one candidate). The converse does not hold, however. Consider for instance the VPT in Figure 4.8 with plain arrows. It encodes a unary tree, so it is in HBM as explained above. Moreover, it is not deterministic, due to the decision on the last letter r vs r'.

This means that there is a strict hierarchy:

$$\text{BM} \subsetneq \text{determinizable VPTs} \subsetneq \text{HBM} \subsetneq \text{functional VPTs}$$

For the first inclusion, recall that BM is also the set of functions definable by a deterministic one-way transducer (by Proposition 4.2 and Proposition 4.4).

Horizontal twinning property. For functional one-way transducers, BM is captured by sequentiality, and this can be decided through a twinning property in PTime [WK95, BC02, BCPS03]. We define a variant of this property in order to capture HBM functions.

A functional VPT $\mathcal{T}$ satisfies the *horizontal twinning property* (HTP) if for all $u_1, u_2 \in \Sigma^*$ such that u_2 is well-nested, for all $v_1, v_2, w_1, w_2 \in \Delta^*$, for all $q_0, q_0' \in I$, for all $q, q' \in Q$, and for all $\sigma, \sigma' \in \Gamma^*$ such that (q, σ) and (q', σ') are co-accessible,

$$\text{if } \begin{cases} (q_0, \bot) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \\ (q_0', \bot) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \end{cases} \quad \text{then } \mathsf{del}(v_1, w_1) = \mathsf{del}(v_1 v_2, w_1 w_2).$$

Recall that $\mathsf{del}(u, v)$ is the delay between u and v, as defined on page 19. Intuitively, the HTP ensures that, on two runs on the same input, the delays cannot increase when traversing well-nested words (i.e. "moving horizontally"). This suffices to capture exactly HBM functions.

Theorem 4.5. *Let $\mathcal{T}$ be a functional VPT.*

1. *$[\![\mathcal{T}]\!]$ is HBM iff $\mathcal{T}$ satisfies the HTP,*
2. *this is decidable in* CoNPTime*, and*
3. *in this case, the algorithm presented in Section 4.2.3 runs in space complexity exponential in the height of the input word.*

Let us give some insights on the proofs. For (1), assume that $\mathcal{T}$ satisfies the HTP. The proof is by induction on the length of the input u' read so far, in a manner similar to [BC02]. The main difference is the additional case where a well-nested factor has a width (when seen as a tree) greater than $|Q|^2$, where we exploit the HTP in full generality. Conversely, if $\mathcal{T}$ does not satisfy the HTP, then we can show that $\mathrm{FST}(\mathcal{T}, K)$ does not satisfy the twinning property for some bound K on the heights of the input words, which means (by Proposition 4.2), that $[\![\mathrm{FST}(\mathcal{T}, K)]\!]$ is not BM, and thus $\mathcal{T}$ not HBM (otherwise it would be bounded by $f(\mathsf{h}(u))$ for some f, and thus bounded on all words of height at most K).

For (2), deciding HTP on a VPT $\mathcal{T}$ is reduced to the emptiness of a pushdown automaton with 2 counters making at most 1 reversal (increasing and then decreasing), which is decidable in CoNPTime [FRR+18]. Indeed, not satisfying the HTP is only possible by two ways: either $|v_2| \neq |w_2|$ (in the premises of the HTP), or $v_2 w_2 \neq \epsilon$ and $v_2[i] \neq w_2[i]$ for some i. Each condition can be checked using two counters and one reversal. The stack is used to check that u_2 is well-matched.

The space complexity of (3) is derived from the proof of (1).

Note that this exponential bound is tight. Consider for instance the function that maps $f(t, a)$ to $f(t, a)$ and $f(t, b)$ to $f(\bar{t}, b)$ where f, a, b are letters from $\Sigma = \Delta$, t is the (depth-first,

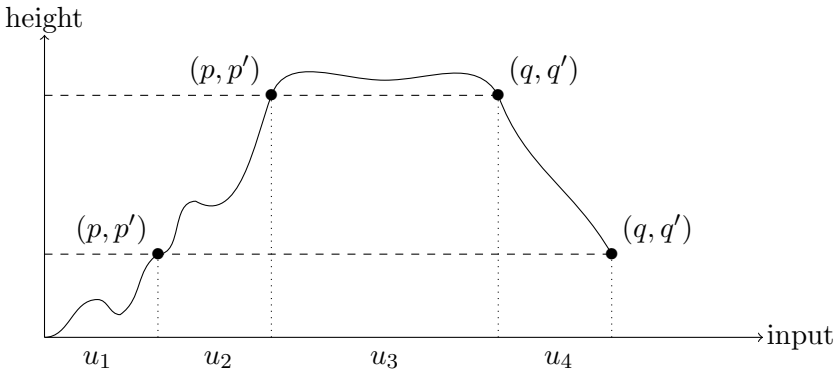


Figure 4.9: Premisses of the matched twinning property (MTP).

left-to-right) encoding of a binary tree over $\{0, 1\}$, and $\bar{t}$ is the encoding of the complement of t, i.e. 0s are replaced by 1s, and conversely. This function is in HBM as this is the encoding of a ranked tree. However it cannot be evaluated with a Turing transducer with a polynomial amount of space because there is a doubly exponential number of such binary trees of a given height, and such a tree has to be stored (or, at least, identified).

4.2.5 Online-bounded memory

While height-bounded memory allows an evaluation with a memory that do not depend on the length of the input, but only on its height, this can be considered as too permissive. This is witnessed for instance by the fact that, if a function operates on encodings of ranked trees, it can store the whole input in memory, and remains in HBM. We define a third class of functions, similar to HBM, but where the amount of space is bounded by the *current* height $\mathsf{hc}(u')$ of the prefix u' of u at any time, not the *global* height $\mathsf{h}(u)$.

A function $f : \Sigma^* \to \Delta^*$ is *online-bounded memory* (OBM) if there exists a Turing transducer M computing it, and a function $\theta : \mathbb{N} \to \mathbb{N}$ such that, just after processing a prefix u' of a well-nested word $u \in \Sigma^*$, the memory used in the working tape of M is less than $\theta(\mathsf{hc}(u'))$.

For instance, consider the function that maps $c^n r^n$ to $a^n c^n$ and $c^n r' r^{n-1}$ to $b^n c^n$ (for any $n > 0$). We claim that this function is OBM, thanks to the following algorithm. First, the number of c's is stored when reading c^n, and until the first return symbol is read. This costs $\log(n)$, and at this time point n is the current height. If the first return symbol is r, it outputs $a^n c$, otherwise $b^n c$. Then the memory is flushed, and a c is emitted each time an r is read.

Functions definable by deterministic VPTs are clearly OBM. The converse is still not true: The previous example is OBM, but cannot be implemented with a deterministic VPT. Thus we obtain the following inclusions:

$$\text{BM} \subsetneq \text{determinizable VPTs} \subsetneq \text{OBM} \subsetneq \text{HBM} \subsetneq \text{functional VPTs}$$

The inclusion OBM $\subset$ HBM is by definition. Its strictness is illustrated for instance by the VPT in Figure 4.8 including only plain arrows. As we have seen, it defines a function in HBM (it only involves ranked trees), but is not in OBM, as the whole input needs to be stored until the last letter.

Matched twinning property. In order to decide whether the function defined by a VPT is OBM, we follow the same approach as for HBM, by defining an appropriate twinning property (see Figure 4.9).

A functional VPT $\mathcal{T}$ satisfies the *matched twinning property* (MTP) if for all $u_i \in \Sigma^*$ and $v_i, w_i \in \Delta^*$ ($i \in \{1, \dots, 4\}$) such that u_3 is well-nested and u_2u_4 is well-nested and, for all $i, i' \in I$, for all $p, q, p', q' \in Q$, and for all $\sigma_1, \sigma_2 \in \Gamma^*$, for all $\sigma_1', \sigma_2' \in \Gamma^*$, such that (q, σ_1) and (q', σ_2) are co-accessible:

$$\text{if} \quad \begin{cases} (i, \bot) \xrightarrow{u_1/v_1} (p, \sigma_1) \xrightarrow{u_2/v_2} (p, \sigma_1\sigma_1') \xrightarrow{u_3/v_3} (q, \sigma_1\sigma_1') \xrightarrow{u_4/v_4} (q, \sigma_1) \\ (i', \bot) \xrightarrow{u_1/w_1} (p', \sigma_2) \xrightarrow{u_2/w_2} (p', \sigma_2\sigma_2') \xrightarrow{u_3/w_3} (q', \sigma_2\sigma_2') \xrightarrow{u_4/w_4} (q', \sigma_2) \end{cases}$$

then $\mathsf{del}(v_1v_3, w_1w_3) = \mathsf{del}(v_1v_2v_3v_4, w_1w_2w_3w_4)$.

Observe that if a VPT satisfies the MTP, it also satisfies the HTP, by taking $u_3 = u_4 = \epsilon$. While the HTP allows to pump "horizontally", the MTP also allows to pump "vertically".

Theorem 4.6. *Let $\mathcal{T}$ be a functional VPT.*

1. *$[\![\mathcal{T}]\!]$ is OBM iff $\mathcal{T}$ satisfies the MTP,*
2. *this is decidable in* CoNPTime*, and*
3. *if this is the case, $[\![\mathcal{T}]\!]$ is computable by a Turing transducer using quadratic space in the current height of the input.*

The proofs follow the same lines as for HBM (cf Theorem 4.5), with more involved developments. For instance, we use a recent result by Aleksi Saarela [Saa19] on word combinatorics, in order to derive more delays when the conditions of the MTP do not hold.

Chapter 5

Algebraic characterizations

Contents

In the previous chapters, we mainly focused on a "machine" approach to transductions, through two-way transducers or streaming string transducers (with a short excursion to logic). In this chapter, we adopt the "algebraic" and "logic" point of view on transductions, still focusing on definability problems.

Languages. For languages, this shift from automata to algebra and logic has been established in the early days of formal language theory, in several ways. For instance the syntactic congruence provides an algebraic tool to characterize regular languages: the Myhill-Nerode theorem states that a language is regular iff its syntactic right congruence has finite index [Ner63], or equivalently iff it is recognized by a finite monoid [Myh57]. Moreover this congruence defines the minimal deterministic automaton recognizing the language. Another well-known connection has been established, this time between logic and automata: a language is regular iff it is recognized by an MSO formula [Büc60, Tra61].

Such correspondences have also been proven for subclasses of regular languages [Str94], for instance star-free languages. These languages correspond to counter-free automata, aperiodic (finite) syntactic congruences, and to languages defined by first-order formulas [Sch65, MP71]. Self-contained proofs of these relations are available in [DG08] (and [DGK08] for classes below).[1] In this chapter we will describe other congruence classes with several characterizations.

[1] Another class is the extension of first-order logic with deterministic transitive closure, captured by two-way multi-head deterministic automata with nested pebbles [EH07].

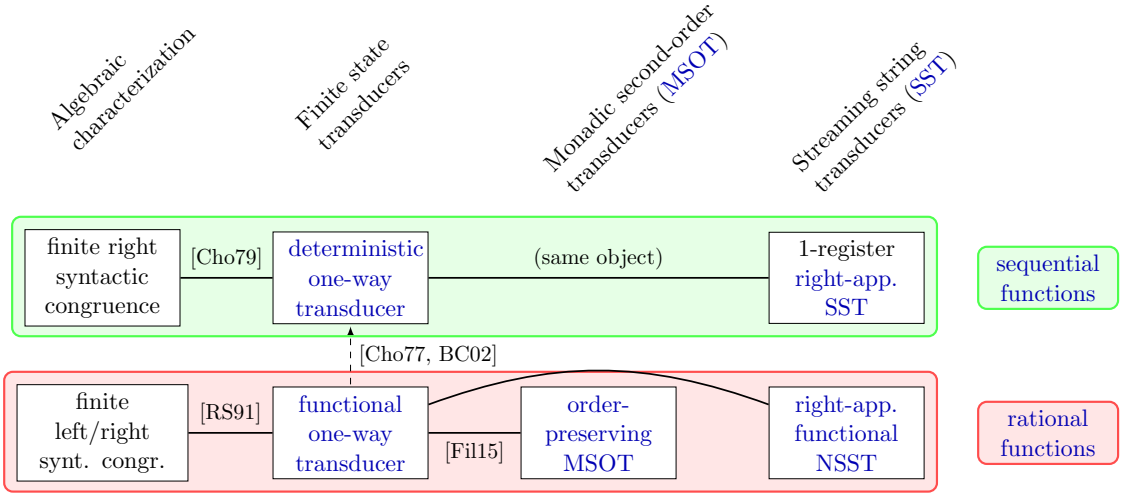


Figure 5.1: Models of finite word to finite word transformations, including algebraic models.

For transductions, the situation is more complex, and almost nothing is known beyond rational functions [Ber79]. We characterize transductions through congruence classes $\mathbf{C}$, (as for instance the class $\mathbf{A}$ of aperiodic congruences): a transduction is $\mathbf{C}$-rational if it can be implemented by a transducer such that the transition congruence of its underlying automaton is in $\mathbf{C}$. Our goal here, is to decide whether a transduction is $\mathbf{C}$-rational, given a decidable congruence class $\mathbf{C}$. In Figure 5.1, we extend Figure 2.1 on transductions classes by adding algebraic characterizations, for sequential and rational functions. Let us depict the situation.

Sequential transductions. Sequential functions, i.e. functions definable by deterministic one-way transducers[2], benefit from a minimization procedure that yields a canonical transducer. Its transition congruence can be defined in a way similar to the language case, once outputs have been moved "to the left" on transitions (that is, outputs are produced as early as possible) [Cho79, Cho03, BC02]. This leads to an algorithm deciding, given a congruence class $\mathbf{C}$ and a sequential transducer, whether there exists an equivalent transducer which is and deterministic and has a transition congruence in $\mathbf{C}$. However, this is not sufficient to decide whether the corresponding sequential function is $\mathbf{C}$-rational: there may be equivalent transducers in $\mathbf{C}$, but none of them is deterministic.

Rational functions. Algebraic characterizations, when operating on automata, need deterministic devices. An early result by Elgot and Mezei shows that each rational transduction is the composition of a co-deterministic transducer (that is, deterministic while reading from right to left) with a deterministic transducer [EM65]. Schützenberger defined the notion of bimachine, that can be considered as deterministic transducers with a co-deterministic regular look-ahead [Sch61][3]. A notion of canonical bimachine has been proposed in [RS91], but this one is not minimal in the algebraic sense: A transduction can have a bimachine with congruences in a congruence class $\mathbf{C}$, but not its canonical bimachine.

[2]Recall that we associate outputs to transitions of a run, but also to its first and last states. These were often named subsequential functions [Sch77].

[3]The name bimachine has been proposed by Eilenberg [Eil74].

Infinite words. For infinite words, some logic-automata connections still hold, but the situation is more intricate on the algebraic side, already at the "language" level [Wil93, PP04]. For instance there is no Myhill-Nerode theorem (i.e. no unique minimal automaton) nor even a canonical automaton. Still, it has been shown in [BC04] how to decide whether a rational function on infinite words is sequential[4] (realized by a transducer with a deterministic underlying Muller automaton). In [Wil16], Wilke describes a notion of bimachine for infinite words, restricted to total letter-to-letter rational functions. A connection between counter-free letter-to-letter bimachines and temporal logics is established. These bimachines over infinite words use the notion of *prophetic* automaton as right automaton (or regular look-ahead) as proposed by Carton and Michel [CM03, Car10]. Also, the equivalence between star-free expressions and aperiodic languages holds for infinite words [Per84], based on a syntactic congruence defined in [Arn85], which leads to the decidability of first-order definability for regular languages of infinite words. The connection between aperiodic two-way transducers and aperiodic SSTs over infinite words is established in [DKT16].

Contributions. In the first part of the chapter, we propose an algorithm for deciding whether a rational transduction on *finite words* is **C**-rational, given a decidable congruence class **C**. This is obtained by proving that there is only a finite number of minimal bimachines, and by finding a way to enumerate them. We also consider the special case of aperiodic congruences, for which we prove that the canonical bimachine is aperiodic iff the transduction is. Furthermore, we establish a transfer theorem between algebra and logic, which is used to extend the results on languages to transductions, in particular the decidability of being definable in first-order logic, but also in other logics having an algebraic counterpart.

The second part of the chapter deals with the extension to *infinite words*. We first show how deterministic transducers over infinite words can be minimized. Then, we extend the notion of *bimachines* to infinite words, and show that this captures the class of rational functions. We define two right congruences (or equivalently, regular look-aheads), that capture enough information from the suffix of the input word in order to have a bimachine recognizing the function, and coarse enough to obtain an aperiodic bimachine, when the function is. This way, we obtain a *canonical* bimachine for any rational function over infinite words, and also a decision procedure for deciding whether such a function is aperiodic. We also show that aperiodic functions correspond to those definable in first-order logic on infinite words. As a side result, we show that the result by Elgot and Mezei stating that a rational function over finite words is the composition of a deterministic and a co-deterministic rational function, also holds for infinite words, in both directions (one of them was proved in [Car10]).

Further related work. In this chapter, we classify functions based on congruence classes, where congruences are those of the underlying automata of the transducers (or bimachines). An alternative way of classifying functions is to use the notion of *continuity* of a class of languages: the function belongs to a given class if it preserves this class by inverse image. This is explored in [CKLP15, CCP17]. For *length-preserving* rational functions, it was proved in [LMSV99] that aperiodic non-deterministic[5] rational functions correspond to length-preserving first-order definable functions. This result has then been generalized to other varieties [MSTV06]. *Non-deterministic bimachines* are studied in [SY06], in terms of expressiveness, not from an algebraic point of view. Regular functions have no algebraic characterization so far, but it has been proved

[4] for functions mapping infinite words to *infinite* words, while in our setting we need functions mapping infinite words to *finite or infinite* words.

[5] Here, "aperiodic non-deterministic" means that the automaton obtained after minimization of the underlying automaton is aperiodic.

that first-order definable regular functions are those definable by aperiodic streaming string transducers [FKT14], and also by aperiodic two-way transducers [CD15] (see also [DJR18] for direct translations between these models). Still, deciding whether a regular function is first-order definable is open.

Open problem 4 (*Regular functions in* FOT)

Is it decidable whether a regular function (given, for instance, by a functional two-way transducer) is definable in FOT*?*

The particular case where input and output alphabets are unary is treated in [CG14], and rotating and sweeping transducers in [Gui16b] (see also [Gui18] for an overview). Biautomata [KP12] share some similarities with bimachines, but are different devices, as their left and right heads share information on the current state.

5.1 Rational functions over finite words

5.1.1 Congruences for transductions

Words. Let us define the *prefix distance* between two words u and v as $\|u, v\| = |u|+|v|-2|u\wedge v|$ (recall that $u\wedge v$ stands for the longest common prefix of u and v).

Automata. Given a one-way automaton $\mathcal{A}$, we note $q \xrightarrow{u}_{\mathcal{A}} q'$ whenever there is a run of $\mathcal{A}$ on u from a state q to a state q'. We omit $\mathcal{A}$ when it is clear from the context. A state q of $\mathcal{A}$ is said *accessible* if there exists a word u and an initial state q_0 of $\mathcal{A}$ such that $q_0 \xrightarrow{u}_{\mathcal{A}} q$. An automaton is said *accessible* if all its states are accessible. **In this chapter we always assume automata to be one-way, deterministic, and complete** (i.e. for every state p and letter a, there is a state q and a transition rule $p \xrightarrow{a}_{\mathcal{A}} q$). Any one-way automaton can be made complete in PTime.

Transducers. We take in this chapter a definition of transducers that slightly differs from the one used in the other chapters, allowing initial outputs. A (one-way) transducer $\mathcal{T}$ is a tuple $(\mathcal{A}, \mathsf{out}, \imath, \jmath)$ where $\mathcal{A} = (Q, \Sigma, \vdash, \dashv, \delta, I, F)$ is still a one-way automaton, out is still an output function $\mathsf{out} : \delta \to \Delta^*$, and where two output functions are added for the initial and final states: $\imath : I \to \Delta^*$ and $\jmath : F \to \Delta^*$.

Given a word $u \in \Sigma^*$ and a run $q_0 q_1 \cdots q_{|u|}$ of $\mathcal{A}$ on u, we write $q_0 \xrightarrow{u|v}_{\mathcal{T}} q_{|u|}$ to denote the corresponding run of $\mathcal{T}$ on u, where $v = \mathsf{out}(q_0, u[1], q_1) \cdots \mathsf{out}(q_{|u|-1}, u[|u|], q_{|u|})$. If $q_0 \in I$ and $q_{|u|} \in F$ then the run is accepting, and $(u, w) \in [\![\mathcal{T}]\!]$ with $w = \imath(q_0) v \jmath(q_{|u|})$ (also written as $[\![\mathcal{T}]\!](u) = w$, as $\mathcal{T}$ is functional). As usual, a transducer is said unambiguous (resp. deterministic) if its underlying automaton is unambiguous (resp. deterministic), and a function is said rational (resp. *sequential*) if it is realized by a functional (resp. deterministic) transducer. Let us recall that functionality is decidable in PTime and that every functional transducer has an equivalent unambiguous transducer [Ber79]. Because of the presence of the output functions $\imath$ and $\jmath$, our notion of deterministic transducers is often called "subsequential" in the literature [Sch77].

Congruences. Let $\sim$ denote an equivalence relation on Σ^*, and $[u]_\sim$ the equivalence class of the word $u \in \Sigma^*$ (also written $[u]$ when it is clear from the context). The relation $\sim$ has *finite index* when its *quotient* $\Sigma^*/_\sim = \{[u]_\sim \mid u \in \Sigma^*\}$ is finite.

Given two equivalence relations $\sim_1$, $\sim_2$ over Σ^*, the relation $\sim_1$ is said *finer* than $\sim_2$ (or $\sim_2$ is *coarser* than $\sim_1$) if for every pair of words u, v, if $u \sim_1 v$ then $u \sim_2 v$. We denote this fact by writing $\sim_1 \sqsubseteq \sim_2$.

A *right congruence* (resp. *left congruence*) over Σ^* is an equivalence relation over Σ^* such that, for every pair of words $u, v \in \Sigma^*$, and every letter $a \in \Sigma$, if $u \sim v$ then $ua \sim va$ (resp. $au \sim av$). An equivalence relation is a *congruence* if it is both a left congruence and a right congruence. The *intersection* of two right (resp. left) congruences $\sim_1, \sim_2$ over a common alphabet is also a right (resp. left) congruence that we denote by $\sim_1 \sqcap \sim_2$.

We say that a congruence $\sim$ *recognizes* a language $L \subseteq \Sigma^*$ if L is a union of equivalence classes C of $\sim$, that is: $L = \bigcup_{c \in C} \{u \mid [u] = c\}$.

Syntactic congruence. One of the most fundamental congruences associated with a language L is its *syntactic congruence* $\equiv_L$ defined by:

$$u \equiv_L v \iff (\forall w_\ell, w_r \in \Sigma^*,\ w_\ell \cdot u \cdot w_r \in L \iff w_\ell \cdot v \cdot w_r \in L)$$

The relation $\equiv_L$ is indeed a congruence, and recognizes L. More importantly, it is the coarsest congruence among all those recognizing L. One can even show (using the transition congruence below) that a language is regular iff its syntactic congruence has finite index.

Transition congruence. Let us now consider two congruences associated with an automaton $\mathcal{A}$ with states Q (recall that $\mathcal{A}$ is assumed one-way, deterministic and complete).

The *transition congruence* $\approx_{\mathcal{A}}$ of $\mathcal{A}$ is defined by:

$$u \approx_{\mathcal{A}} v \iff (\forall p, q \in Q,\ p \xrightarrow{u}_{\mathcal{A}} q \iff p \xrightarrow{v}_{\mathcal{A}} q)$$

The relation $\approx_{\mathcal{A}}$ is a congruence. Let L denote the language recognized by $\mathcal{A}$: in this case $\approx_{\mathcal{A}}$ recognizes L.

The *right transition congruence* $\sim_{\mathcal{A}}$ of $\mathcal{A}$ (with initial state q_0) is defined by:

$$u \sim_{\mathcal{A}} v \iff (\forall q \in Q,\ q_0 \xrightarrow{u}_{\mathcal{A}} q \iff q_0 \xrightarrow{v}_{\mathcal{A}} q)$$

This relation is a right congruence. We can see in this definition that an equivalence class of $\sim_{\mathcal{A}}$ is strongly related with the state q reached when reading a member of the class. Hence we will often identify a state q and the class $[u]_{\sim_{\mathcal{A}}}$ of words u leading to q, and write $[u]_{\mathcal{A}}$ for simplicity. We also use $\sim_{\mathcal{A}}$ to compare automata: given two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, we say that $\mathcal{A}_1$ is finer than $\mathcal{A}_2$ (written $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$) iff $\sim_{\mathcal{A}_1} \sqsubseteq \sim_{\mathcal{A}_2}$. With this definition, the minimal deterministic automaton recognizing L is also the coarsest deterministic automaton recognizing L (up to isomorphism).

Congruence classes. Given an alphabet Σ, a *congruence class* $\mathbf{C}$ is a set of congruences of finite index over Σ which is both: 1) closed under intersection, and 2) closed under taking coarser congruences.[6]

In the following we will consider definability problems related to congruences, so we need to agree on a finite representation of them. A congruence $\sim$ of finite index over Σ can be fully described by a morphism $\mu : \Sigma^* \to M$, where M is a finite monoid, and μ is such that $u \sim v$

[6]The notion of recognizability by a congruence is equivalent to the notion of recognizability by a *stamp* (surjective morphism from a free monoid to a finite monoid). However the notion of variety of stamps defined in [PS05] differs slightly from our notion of congruence class (such a variety is always a congruence class in our setting, but not the converse).

iff $\mu(u) = \mu(v)$. This morphism can itself be fully described by a function $f : \Sigma \to M$. From now on, we always assume that a congruence is described by such a monoid and morphism. A congruence class $\mathbf{C}$ is said *decidable* if the following problem is decidable: given a congruence $\sim$ (described by a monoid and morphism), decide if $\sim$ is in $\mathbf{C}$.

In the sequel we may refer to the following decidable congruence classes:

- the class $\mathbf{F}$ of all finite congruences,
- the class $\mathbf{I}$ of idempotent congruences: a congruence $\sim$ is *idempotent* if for every word u, $u \sim u^2$,
- the class $\mathbf{A}$ of aperiodic congruences: a congruence $\sim$ is *aperiodic* if there exists n such that, for every word u, $u^n \sim u^{n+1}$.
- the class $\mathbf{DA}$ which gathers the languages definable in first-order logic with two variables [TW98]. A congruence $\sim$ is in $\mathbf{DA}$ if there exists n such that for any words u, v, w, we have: $(uvw)^n v (uvw)^n \sim (uvw)^n$.
- the class $\mathbf{J}$ of $\mathcal{J}$-trivial congruences. The Green relation $\mathcal{J}$ is defined by: $u \mathcal{J} v$ whenever there exists u', u'', v', v'' such that $u \sim v'vv''$ and $v \sim u'uu''$. A congruence $\sim$ is *$\mathcal{J}$-trivial* if for all words u, v in Σ^*, if $u \mathcal{J} v$ then $u = v$.

C-automata and C-transducers. Given a congruence class $\mathbf{C}$, we will say that an automaton $\mathcal{A}$ is a ***C*-automaton** if its transition congruence $\approx_{\mathcal{A}}$ is in $\mathbf{C}$. A ***C*-transducer** is a transducer whose underlying automaton is a $\mathbf{C}$-automaton. We say that a function is ***C*-rational** (resp. ***C*-sequential**) if it is realized by a functional (resp. deterministic) $\mathbf{C}$-transducer.

Definability. Let us now focus on the following definability problem:

> Given a functional one-way transducer recognizing a function f, and a congruence class $\mathbf{C}$, is f $\mathbf{C}$-rational?

We always assume here, that $\mathbf{C}$ is decidable. We will first focus on the subset of sequential functions, for which the problem is simpler to address.

5.1.2 Sequential functions

The situation for sequential functions is somehow similar to that of regular languages, with an additional subtlety.[7] Given a regular language L and a congruence class $\mathbf{C}$, one can decide if there exists a congruence in $\mathbf{C}$ recognizing L by taking a deterministic automaton recognizing L, minimizing it, and testing if this automaton has a transition congruence in $\mathbf{C}$.

For sequential functions, Choffrut provided a minimization algorithm [Cho03], that we describe hereafter. As for languages, using this minimization procedure on a deterministic transducer recognizing a function f, and testing if this new transducer is in $\mathbf{C}$, permits to decide if f is $\mathbf{C}$-sequential:

Theorem 5.1. *Let $\mathbf{C}$ be a decidable congruence class. It is decidable whether a sequential function, given by a one-way transducer, is **C**-sequential.*

[7] Recall that what we name "sequential" functions here are usually named "subsequential", as they can output an additional word at the beginning (resp. end) of a run, depending on the initial (resp. final) state.

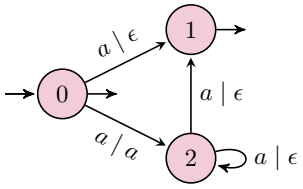


Figure 5.2: An **I**-transducer.

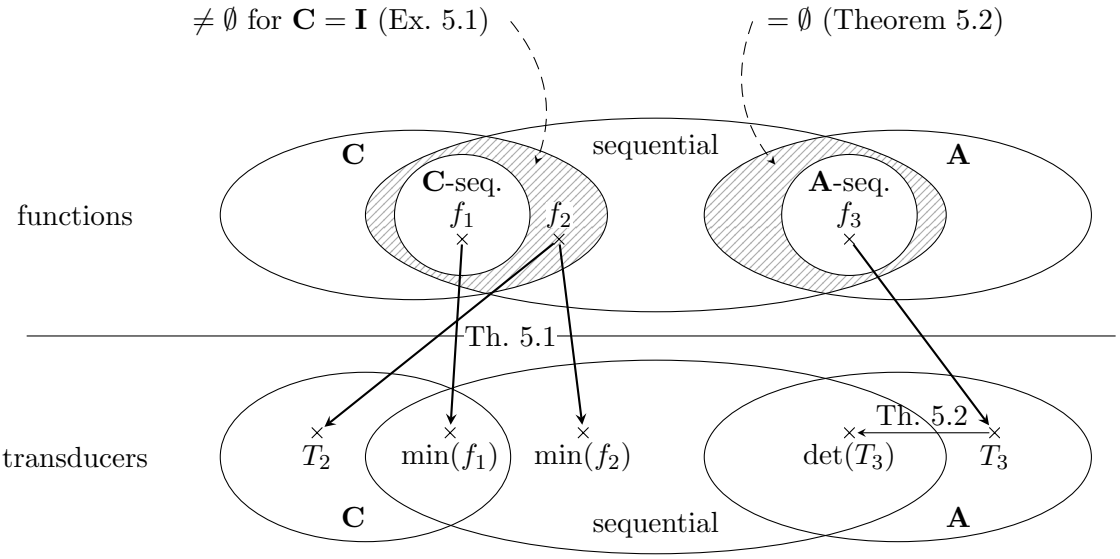


Figure 5.3: Situation for sequential transductions.

C + sequential ≠ C-sequential. The subtlety is that it does not permit to decide if f is C-rational. Indeed, being C-rational and sequential is not sufficient for being C-sequential. In other terms, there are classes **C** and functions f that are C-rational (there is a transducer in **C** recognizing f), sequential (there is a deterministic transducer recognizing f), but are not C-sequential (i.e. there is no transducer recognizing f which is *both* in **C** *and* sequential). Let us exhibit such a class and function.

Example 5.1. *Consider the function f over alphabets $\Sigma = \Delta = \{a\}$ such that $f(\epsilon) = f(a) = \epsilon$ and $f(a^n) = a$ for all $n > 1$. Clearly, this function is sequential, with a transducer that outputs a on the second input letter, and ϵ on all others.*

*As a class, we consider the class **I** of idempotent congruences defined before. We provide in Figure 5.2 a (non-deterministic) **I**-transducer recognizing f.*

*Now, we claim that there is no deterministic **I**-transducer recognizing f. Assume for contradiction that such a transducer exists, and let $p \xrightarrow{a|\epsilon} q$ be its accepting run over a. We have $i(p) = t(p) = \epsilon$ since $f(a) = \epsilon$. As $f(aa) = a$ and $a \sim aa$ ($\sim$ is idempotent), we have $q \xrightarrow{a|a} q$, and thus $f(aaa) = aa$, which is a contradiction.*

The global picture is depicted in Figure 5.3, and for now we described the leftmost part. For the class **A** of aperiodic congruences (on the right part), the situation is a bit simpler:

Theorem 5.2. *A sequential function is **A**-sequential if and only if it is **A**-rational.*

This property is proved by checking that the determinization algorithm in [BC02] preserves aperiodicity. Together with Theorem 5.1, we obtain a procedure to decide **A**-sequential and **A**-rationality of sequential functions.

Minimization. Let us give some intuitions about why Theorem 5.1 holds. The minimization procedure proposed by Choffrut [Cho03] is based on the following ingredients, defined from a function $f : \Sigma^* \to \Delta^*$:

- a new function $\widehat{f} : \Sigma^* \to \Delta^*$ defined by:

$$\widehat{f}(u) = \bigwedge \{f(uw) \mid w \in u^{-1}\mathrm{dom}(f)\}$$

  $\widehat{f}(u)$ outputs the longest common prefix of all the possible continuations of u in the domain, which is the maximal output that can be produced, and this is done at any time point.

- the *syntactic congruence* of f, denoted by $\sim_f$, and defined by:

$$u \sim_f v \text{ iff } \begin{array}{l} 1) \text{ for any } w \in \Sigma^*,\ uw \in \mathrm{dom}(f) \iff vw \in \mathrm{dom}(f) \text{ and} \\ 2) \text{ for any } w \in u^{-1}\mathrm{dom}(f),\ \widehat{f}(u)^{-1}f(uw) = \widehat{f}(v)^{-1}f(vw), \text{for any words } u, v. \end{array}$$

  Condition 1) ensures that $\sim_f$ recognizes $\mathrm{dom}(f)$, while Condition 2) checks that the output for any w is the same after reading u and v, once their specific outputs $\widehat{f}(u)$ and $\widehat{f}(v)$ have been removed.

The minimal transducer $\mathcal{T}_f$ is defined from $\sim_f$ in a natural way:

- its underlying automaton has the classes of $\sim_f$ as states, with initial state $[\epsilon]$, final states $\{[u] \mid u \in \mathrm{dom}(f)\}$, and transition rules $[u] \xrightarrow{a} [ua]$ for all $u \in \Sigma^*$ and $a \in \Sigma$.
- the outputs are defined by: $\mathsf{out}([u] \xrightarrow{a} [ua]) = \widehat{f}(u)^{-1}\widehat{f}(ua)$, which means that when reading a letter a after u, $\mathcal{T}_f$ outputs the maximal output for ua, given that it has already output the maximal output for u. For the initial and final states we take $\imath([\epsilon]) = \widehat{f}(\epsilon)$ and $\jmath([u]) = \widehat{f}(u)^{-1}f(u)$, for $u \in \mathrm{dom}(f)$.

With this definition, f is **C**-sequential iff $\mathcal{T}_f$ is a **C**-transducer. Moreover, $\mathcal{T}_f$ can be computed in PTime from any sequential transducer realizing f [Cho03]. This completes the sketch of proof for Theorem 5.1.

5.1.3 Bimachines

Having a *deterministic* device is a major feature for establishing algebraic properties. This is the reason why the result for sequential functions was easily obtained from the minimization procedure of deterministic transducers, and why regular languages (that can all be described by deterministic automata) have a simple algebraic characterization, by finite congruences. However, no simple deterministic device captures rational functions, so their algebraic characterization is more challenging. Still, Elgot and Mezei showed that any rational function is the composition of a left-to-right sequential transducer, and a sequential right-to-left transducer [EM65, Section 7].

Schützenberger proposed the model of bimachines in [Sch61], which was also developed (and named) later by Eilenberg [Eil74]. A bimachine $\mathcal{B}$ is composed by two automata: a deterministic left-to-right automaton $\mathcal{L}$ (called left automaton), and a deterministic right-to-left automaton $\mathcal{R}$ (called right automaton). An output function gives, from the pair of states of the two automata at a given position, the word output at that position. This would rather be considered nowadays as a deterministic automaton with a regular co-deterministic look-ahead, but the bimachine is a completely symmetrical device. Bimachines are exactly as expressive as functional one-way transducers, i.e. capture rational functions.

Reutenauer and Schützenberger defined a canonical bimachine $\mathcal{B}_f$ associated with any rational function f [RS91]. In this section we show how it is defined. We start from a bimachine $\mathcal{B}$ realizing a function f:

1. one can define a canonical right automaton for this function, i.e. a right automaton $\mathcal{R}$ that depends on f, but not on $\mathcal{B}$. We write it $\mathcal{R}_f$.

2. when the right automaton $\mathcal{R}$ is fixed, one can minimize the left automaton $\mathcal{L}$ of $\mathcal{B}$, that we write $\mathit{Left}(\mathcal{R})$. This is called left-minimization, and yields a new bimachine written $\mathit{Left}(\mathcal{B})$. Symmetrically, we can define the *right-minimization* $\mathit{Right}(\mathcal{B})$.

3. from this, one defines the canonical bimachine $\mathcal{B}_f$ with right automaton $\mathcal{R}_f$, and left automaton $\mathit{Left}(\mathcal{R}_f)$.

A ***C*-bimachine** is a bimachine whose left and right automata are both **C**-automata. We will see (in Section 5.1.4) that unfortunately canonical bimachines do not gather all the algebraic properties of a function, i.e. there exist **C**-rational functions f such that $\mathcal{B}_f$ is not a **C**-bimachine. Hence, we will need another procedure. This procedure heavily relies on the core operations described above: left-minimization and canonical right automaton. For this reason we describe their definitions and basic properties here.

Right automaton, left congruence, bimachine. A *right automaton* $\mathcal{R}$ is a one-way automaton with a single initial state, and with backward deterministic transitions (also called *co-deterministic*). It is interpreted as reading the input word from right to left deterministically. Equivalently, it can also be interpreted as a deterministic automaton operating on the mirror of the word (also swapping initial and final states). The *left transition congruence* $\sim_{\mathcal{R}}$ associated with $\mathcal{R}$ is also defined in the exact symmetric way as transition congruences of one-way automata, and in particular is a left congruence. In the following, we name *left automaton* a deterministic one-way automaton.

A *bimachine* $\mathcal{B}$ is a tuple $(\mathcal{L}, \mathcal{R}, \mathsf{out}, \mathsf{out_l}, \mathsf{out_r})$ where $\mathcal{L}$ and $\mathcal{R}$ are left and right automata with states $Q_{\mathcal{L}}$ and $Q_{\mathcal{R}}$, and final states $F_{\mathcal{L}}$ and $F_{\mathcal{R}}$, respectively. We require that $\mathcal{L}$ and $\mathcal{R}$ have the same domain.[8] The function $\mathsf{out} : Q_{\mathcal{L}} \times \Sigma \times Q_{\mathcal{R}} \to \Sigma^*$ is the *output function*, $\mathsf{out_l} : F_{\mathcal{R}} \to \Sigma^*$ is the *left final function* and $\mathsf{out_r} : F_{\mathcal{L}} \to \Sigma^*$ is the *right final function*. Given two states $l \in Q_{\mathcal{L}}$ and $r \in Q_{\mathcal{R}}$, the output function is extended to words in the following way: $\mathsf{out}(l, \epsilon, r) = \epsilon$, and $\mathsf{out}(l, uv, r) = \mathsf{out}(l, u, r')\mathsf{out}(l', v, r)$, provided that $l \xrightarrow{u}_{\mathcal{L}} l'$ and $r' \xleftarrow{v}_{\mathcal{R}} r$. The function *realized* by $\mathcal{B}$ is the function $[\![\mathcal{B}]\!]$ with the same domain as $\mathcal{L}$ and $\mathcal{R}$, and such that $[\![\mathcal{B}]\!](u) = \mathsf{out_l}(r)\mathsf{out}(l_0, u, r_0)\mathsf{out_r}(l)$ where $l_0 \xrightarrow{u}_{\mathcal{L}} l$ and $r \xleftarrow{u}_{\mathcal{R}} r_0$ are accepting runs of $\mathcal{L}$ and $\mathcal{R}$, respectively. Intuitively, out can be read as a function reading an input letter annotated with the states of $\mathcal{L}$ and $\mathcal{R}$, and producing an output at that position accordingly. The global output on u is then the concatenation of the outputs of each letter of u.

Example 5.2. *The bimachine depicted in Figure 5.4 swaps the first and last letter of words in* $\{a, b\}^*$. *The left automaton stores the first letter, and the right automaton the last one. The output is defined, for every* $c \in \{a, b\}$, *by* $\mathsf{out}(l_0, c, r_c) = \mathsf{out}(l_c, c, r_0) = c$ *and* $\mathsf{out}(l, c, r) = c$ *in all other cases. Moreover* $\mathsf{out_l}(r) = \mathsf{out_r}(l) = \epsilon$ *for all states* l, r.

[8] This requirement was not present in the original paper [RS91]. As a consequence, the results in that paper were limited to *total* functions (see [RS91, Section 5.2]), while we can consider here the usual setting of *partial* functions.

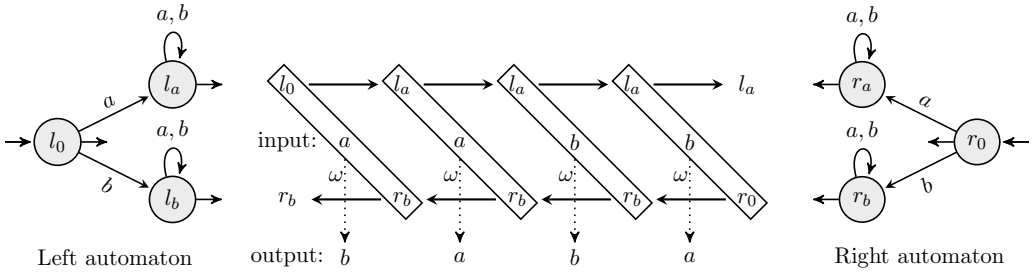


Figure 5.4: Automata of a bimachine $\mathcal{B}$, and a run of $\mathcal{B}$ on the word *aabb*.

Left-minimization. Given two bimachines $\mathcal{B}_1$ and $\mathcal{B}_2$ with respective left automata $\mathcal{L}_1, \mathcal{L}_2$ and right automata $\mathcal{R}_1, \mathcal{R}_2$, we say that $\mathcal{B}_1$ is *finer* than $\mathcal{B}_2$ (or $\mathcal{B}_2$ *coarser* than $\mathcal{B}_1$), denoted $\mathcal{B}_1 \sqsubseteq \mathcal{B}_2$, if $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$ and $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$. A bimachine $\mathcal{B}$ is *minimal* if there is no equivalent bimachine strictly coarser than $\mathcal{B}$ (but other incomparable bimachines may exist).

Consider a bimachine $\mathcal{B}$ realizing a function f, with right automaton $\mathcal{R}$. The *left-minimization* consists in building a minimal[9] left automaton $Left_f(\mathcal{R})$ among all those which, associated with $\mathcal{R}$, define a bimachine realizing f. We usually write $Left(\mathcal{R})$ instead of $Left_f(\mathcal{R})$, for clarity.

Recall that in the sequential case (Section 5.1.2), the minimal deterministic transducer outputs the longest word among all those obtained from a word with the same prefix. The approach here is the same, but on the input word annotated with the equivalence class $[w]_{\mathcal{R}}$ of $\mathcal{R}$ on the current suffix w of the input. We define the functions $\widehat{f}_{[w]_{\mathcal{R}}}$, for any w, by:

$$\widehat{f}_{[w]_{\mathcal{R}}}(u) = \bigwedge \{ f(uv) |\ v \in [w]_{\mathcal{R}} \cap u^{-1}\mathrm{dom}(f) \}$$

which denotes the longest possible output after reading u, given that the suffix v will be in $[w]_{\mathcal{R}}$. Like for sequential functions, we define a right congruence $\sim_L$ from $\widehat{f}_{[w]_{\mathcal{R}}}$, by:

$$u \sim_L v \text{ iff } \begin{cases} \forall w \in \Sigma^*,\ uw \in \mathrm{dom}(f) \Leftrightarrow vw \in \mathrm{dom}(f) \text{ and,} \\ \text{if } uw \in \mathrm{dom}(f),\ \text{then } \widehat{f}_{[w]_{\mathcal{R}}}(u)^{-1} f(uw) = \widehat{f}_{[w]_{\mathcal{R}}}(v)^{-1} f(vw) \end{cases}$$

and the left automaton $Left(\mathcal{R})$ is built from $\sim_L$ in the natural way. Symmetrically, the right automaton $Right(\mathcal{L})$ is defined from a given left automaton $\mathcal{L}$. From a bimachine $\mathcal{B}$, we define the bimachine $Left(\mathcal{B})$ similar to $\mathcal{B}$ except that its left automaton is replaced by $Left(\mathcal{R})$, and output functions are adapted accordingly. $Right(\mathcal{B})$ is defined similarly: the right automaton of $\mathcal{B}$ is replaced by $Right(\mathcal{L})$. The following properties can be proved:

Proposition 5.1. *Given a bimachine $\mathcal{B}$ realizing a function f, with a left automaton $\mathcal{L}$ and a right automaton $\mathcal{R}$,*

1. *$Left(\mathcal{B})$ and $Right(\mathcal{B})$ are well-defined, and equivalent to $\mathcal{B}$,*
2. *$\mathcal{L} \sqsubseteq Left(\mathcal{R})$ and $\mathcal{R} \sqsubseteq Right(\mathcal{L})$,*
3. *$Left(\mathcal{B})$ (resp. $Right(\mathcal{B})$) is minimal among all bimachines equivalent to $\mathcal{B}$ having $\mathcal{R}$ as right automaton (resp. $\mathcal{L}$ as left automaton),*
4. *$Left(Right(\mathcal{B}))$ and $Right(Left(\mathcal{B}))$ are minimal bimachines realizing f. Moreover they are comparable with $\mathcal{B}$, so $\mathcal{B} \sqsubseteq Left(Right(\mathcal{B}))$ and $\mathcal{B} \sqsubseteq Right(Left(\mathcal{B}))$,*
5. *$Left(Right(\mathcal{B}))$ and $Right(Left(\mathcal{B}))$ can be computed in* PTIME *from $\mathcal{B}$.*

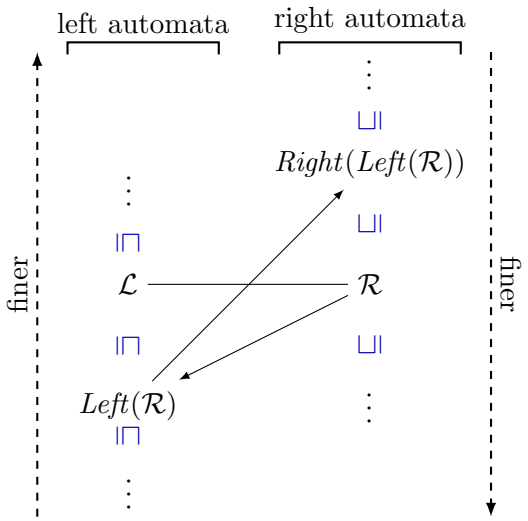


Figure 5.5: Combining left-minimization and right-minimization from a bimachine with automata $\mathcal{L}$ and $\mathcal{R}$.

We represent some of these properties in Figure 5.5, for a given rational function f. The left part represents left automata, and the right part right automata. The line joining $\mathcal{L}$ and $\mathcal{R}$ represents a bimachine $\mathcal{B}$ realizing f, with $\mathcal{L}, \mathcal{R}$ as automata. Left automata are (partially) ordered by $\sqsubseteq$, with the finer left automata at the top. Right automata are ordered in the opposite way, which is natural, as the finer a left automaton is, the coarser an associated right automaton can be (and symmetrically). We can observe for instance that $\mathcal{B} \sqsubseteq \mathit{Right}(\mathit{Left}(\mathcal{B}))$, as $\mathcal{L} \sqsubseteq \mathit{Left}(\mathcal{R})$ and $\mathcal{R} \sqsubseteq \mathit{Right}(\mathit{Left}(\mathcal{L}))$. We use arrows to indicate *Left* and *Right* operations.

We have seen how to define a minimal left automaton, given a fixed right automaton. Let us now look how we can canonically define a right automaton from f.

Canonical right automaton. We define a canonical right automaton: it can be considered as the minimal (co-deterministic) amount of look-ahead information needed in order to have a deterministic left automaton using its information and realizing the function. As expected, the canonical right automaton relies on a left congruence.

The *left congruence of a function* $f : \Sigma^* \to \Delta^*$ is defined by:

$$u \leftharpoonup_f v \text{ if } \begin{cases} \forall w \in \Sigma^*,\ wu \in \mathrm{dom}(f) \Leftrightarrow wv \in \mathrm{dom}(f) \text{ and} \\ \sup\{\|f(wu), f(wv)\| \mid wu \in \mathrm{dom}(f)\} < \infty \end{cases}$$

These conditions state that u and v must have the same prefixes in the domain, and, over any of these prefixes w, f outputs *almost* the same word on wu and wv, i.e. up to a word of bounded length. The relation $\leftharpoonup_f$ is indeed a left congruence, and it is of finite index for any rational function f [RS91]. The *canonical right automaton* of a rational function f, written $\mathcal{R}_f$, is based on $\leftharpoonup_f$ (we write here $[u]$ instead of $[u]_{\leftharpoonup_f}$, for clarity): its set of states is $\Sigma^*/_{\leftharpoonup_f}$ with initial state $[\epsilon]$, final states $\{[u] \mid u \in \mathrm{dom}(f)\}$ and transitions $[au] \xleftarrow{a} [u]$, for all $a \in \Sigma^*$ and $u \in \Sigma^*$. Symmetrically, one can define the *right congruence of a function* $\rightharpoonup_f$ and the *canonical left automaton* of f, written $\mathcal{L}_f$. The canonical right automaton is coarser than any right automaton for f (and symmetrically):

[9] As a deterministic automaton, this can be understood as both "in the number of states", or "with the coarsest transition congruence".

Proposition 5.2. *If $\mathcal{B}$ is a bimachine realizing f, with left automaton $\mathcal{L}$ and right automaton $\mathcal{R}$, then $\mathcal{L} \sqsubseteq \mathcal{L}_f$ and $\mathcal{R} \sqsubseteq \mathcal{R}_f$.*

Canonical bimachine. We have all the ingredients to define the *canonical bimachine* $\mathcal{B}_f$ associated with the rational function f. It is defined as $\mathcal{B}_f = (\mathit{Left}_f(\mathcal{R}_f), \mathcal{R}_f, \mathsf{out}, \mathsf{out}_\mathsf{l}, \mathsf{out}_\mathsf{r})$ where:

$$\begin{aligned} \mathsf{out}([u]_{\sim_L}, \sigma, [w]_{\mathcal{R}_f}) &= \widehat{f}_{[\sigma w]_{\mathcal{R}_f}}(u)^{-1} \widehat{f}_{[w]_{\mathcal{R}_f}}(u\sigma) \\ \mathsf{out}_\mathsf{l}([w]_{\mathcal{R}_f}) &= \widehat{f}_{[w]_{\mathcal{R}_f}}(\epsilon) && \text{for } w \in \mathrm{dom}(f) \\ \mathsf{out}_\mathsf{r}([u]_{\sim_L}) &= \widehat{f}_{[\epsilon]_{\mathcal{R}_f}}(u)^{-1} f(u) && \text{for } u \in \mathrm{dom}(f) \end{aligned}$$

This bimachine is clearly canonical: it is built upon the canonical right automaton $\mathcal{R}_f$, and its left-minimization $\mathit{Left}(\mathcal{R}_f)$, so it does not depend on the initial bimachine (or transducer) realizing f. Reutenauer and Schützenberger showed that $\mathcal{B}_f$ is effectively computable from any transducer or bimachine realizing f [RS91].

5.1.4 Rational functions

C-bimachines vs C-transducers. Recall that a function is **C**-rational if it can be realized by a **C**-transducer, i.e. a one-way transducer with a transition congruence in **C**. This can be lifted to bimachines:

Proposition 5.3. *A function is **C**-rational iff it can be realized by some **C**-bimachine.*

Indeed, from a bimachine $\mathcal{B}$ with left automaton $\mathcal{L}$ and right automaton $\mathcal{R}$, one can build (in PTime) a one-way transducer with a transition congruence coarser than $\approx_\mathcal{B}$ (defined as $\approx_\mathcal{L} \sqcap \approx_\mathcal{R}$) by taking $\mathcal{L} \times \mathcal{R}$ as underlying automaton. Conversely, from a transducer $\mathcal{T}$, one can build (in ExpTime) a bimachine with a transition congruence coarser than that of $\mathcal{T}$, by taking $\approx_\mathcal{T}$ as right automaton $\mathcal{R}$, and $\mathit{Left}(\mathcal{R})$ as left automaton.

Canonicity does not suffice. Now, it is tempting to think that a function f is **C**-rational iff its canonical bimachine $\mathcal{B}_f$ is a **C**-bimachine. Unfortunately, this does not hold for some classes **C**, as for instance the idempotent congruences **I**:

Example 5.3. *Consider the **I**-transducer in Figure 5.2 and the function f it defines, as already described in Example 5.1. We have seen that f is **I**-rational (and sequential, but not **I**-sequential), and we show that $\mathcal{B}_f$ is not a **I**-bimachine. As a sequential function, its canonical right automaton $\mathcal{R}_f$ is the trivial automaton with a single state. Thus $\mathit{Left}(\mathcal{R}_f)$ is the underlying automaton of the minimal deterministic transducer of f. As f is not **I**-sequential, $\mathit{Left}(\mathcal{R}_f)$ is not in **I**, and neither is $\mathcal{B}_f$ (which left automaton is $\mathit{Left}(\mathcal{R}_f)$).*

We will see in Section 5.1.5 that for the class **A** of aperiodic congruences, the canonical bimachine is aperiodic iff the function is.

Decision procedure. At this point, for an arbitrary (decidable) class **C** of congruences, we still do not have an algorithm to decide **C**-rationality. We will manage to obtain it by proving that the number of minimal bimachines[10] is finite (as conjectured in [RS91]).

We have seen in Proposition 5.2 that for any bimachine with automata $\mathcal{L}, \mathcal{R}$ realizing f, $\mathcal{R} \sqsubseteq \mathcal{R}_f$ and $\mathcal{L} \sqsubseteq \mathcal{L}_f$, meaning that canonical left and right automata are $\sqsubseteq$-upper bounds over all left and right automata realizing f. In fact, $\mathit{Left}(\mathcal{R})$ and $\mathit{Right}(\mathcal{L})$ form $\sqsubseteq$-lower bounds, when we consider only minimal bimachines:

[10] up to state equivalence and output shifting, i.e. by identifying $\mathcal{B}_1$ and $\mathcal{B}_2$ once $\mathcal{B}_1 \sqsubseteq \mathcal{B}_2$, $\mathcal{B}_2 \sqsubseteq \mathcal{B}_1$, and they both realize the same function f.

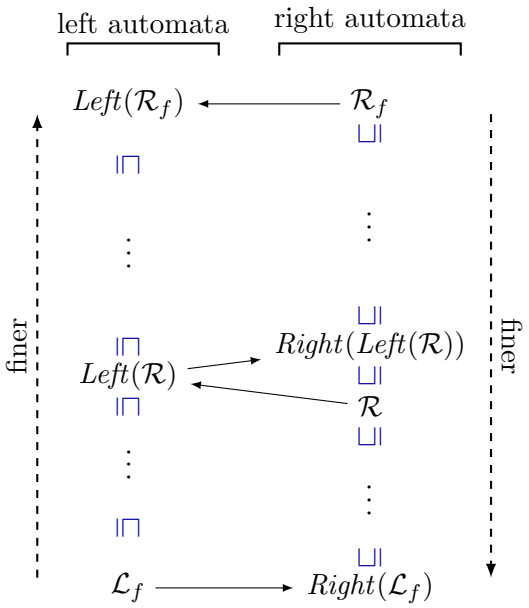


Figure 5.6: A view on *minimal* bimachines of a rational function.

Proposition 5.4. *If $\mathcal{B}$ is a minimal bimachine realizing f with left automaton $\mathcal{L}$ and right automaton $\mathcal{R}$, then $Left(\mathcal{R}_f) \sqsubseteq \mathcal{L}$ and $Right(\mathcal{L}_f) \sqsubseteq \mathcal{R}$.*

In Figure 5.6 we depict *minimal* bimachines in the same way as in Figure 5.5, with the upper and lower bounds for left and right automata. As there is only a finite number of automata coarser than $Left(\mathcal{R}_f)$ and $Right(\mathcal{L}_f)$, respectively, we obtain:

Theorem 5.3. *Given a rational function f, the set of minimal bimachines realizing f is finite (up to equivalence and renaming). One can compute a set of representatives of each class of minimal bimachines, when f is given by a one-way transducer or a bimachine. As a consequence, **C**-rationality is decidable.*

The set of representatives is obtained by computing $Right(\mathcal{L}_f)$ and $\mathcal{R}_f$, and for each $\mathcal{R}$ such that $Right(\mathcal{L}_f) \sqsubseteq \mathcal{R} \sqsubseteq \mathcal{R}_f$, add $Right(Left(\mathcal{B}))$ to the representatives, where $\mathcal{B}$ has $\mathcal{L}, \mathcal{R}$ as automata ($\mathcal{L}$ is arbitrary, it disappears in $Left(\mathcal{B})$). Once the set of representatives is computed, it suffices to check whether one of them is a $\mathbf{C}$-bimachine for deciding **C**-rationality of f.

The situation is illustrated in Figure 5.7. In [FGL19], we define a single bimachine $\mathcal{B}_{f,\mathbf{C}}$ from f, that is in $\mathbf{C}$ iff f is **C**-rational. Its left automaton $\mathcal{L}_{f,\mathbf{C}}$ is obtained by intersecting the congruences of all left automata $\mathcal{L}$ coarser than $Left(\mathcal{R}_f)$, while its right automaton is $Right(\mathcal{L}_{f,\mathbf{C}})$. This provides another way of deciding **C**-rationality.

5.1.5 The aperiodic case

Let us now consider the special case of aperiodic rational functions, i.e. the case where $\mathbf{C} = \mathbf{A}$. Here the situation is simpler:

Theorem 5.4. *If f is aperiodic (i.e. **A**-rational), then so are all its minimal bimachines, and in particular the canonical bimachine $\mathcal{B}_f$.*

As the right automaton of the canonical bimachine $\mathcal{B}_f$ is coarser than any right automaton of a bimachine realizing f (by Proposition 5.2), the right automaton of $\mathcal{B}_f$ is aperiodic, if f is.

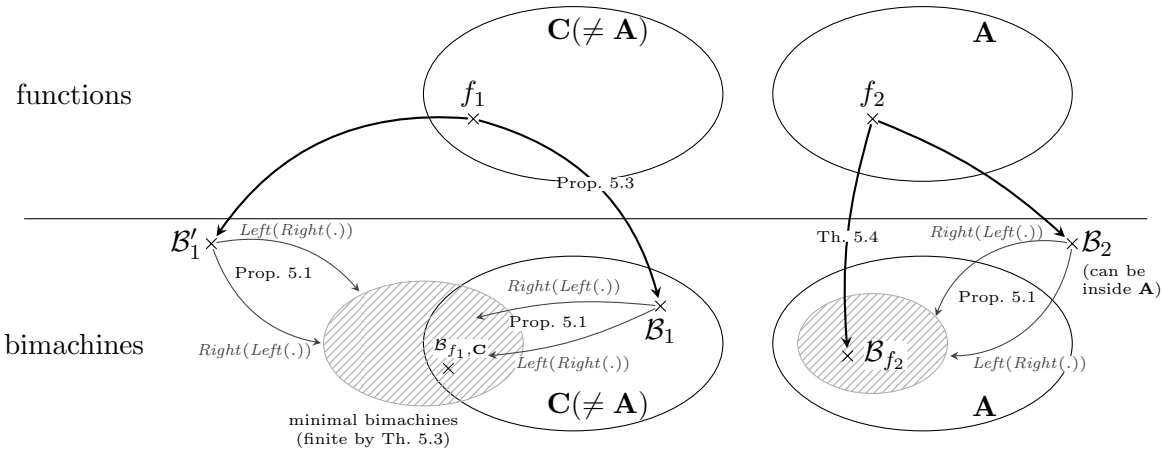


Figure 5.7: Situation for rational transductions.

Proving the same property for the left automaton of $\mathcal{B}_f$ requires a specific development, based on this fact: when f is aperiodic, and viewed as a composition of a deterministic transducer, and a co-deterministic transducer [EM65], then these two transducers are $\mathbf{A}$-sequential. Then, it can be shown that all minimal bimachines are $\mathbf{A}$-bimachines, once f is aperiodic.

Proposition 5.5. *Deciding if a function, given by a bimachine, is aperiodic is* PSPACE*-complete.*

The lower bound comes from the PSPACE-hardness of aperiodicity for languages [CH91]. For the upper bound, computing the minimal bimachine $Left(Right(\mathcal{B}))$ from $\mathcal{B}$ can be done in PTIME, and testing its aperiodicity is in PSPACE [Ste85, CH91]. It is unknown whether the problem remains in PSPACE when starting from a transducer instead of a bimachine:

Open problem 5 (*Aperiodicity of one-way transducers in* PSpace)

Determine whether the following problem is decidable in PSPACE*: Given a one-way transducer realizing a function f, decide if f is aperiodic.*

5.1.6 Logical transducers

We have seen how to decide whether a rational function is $\mathbf{C}$-rational. Let us now study how to lift this algebraic characterization to a logical characterization. This can be achieved through a transfer theorem, which would be ideally[11]:

> Given a fragment $\mathcal{F}$ of MSO equivalent to a congruence class $\mathbf{C}$ of languages, a rational function is $\mathbf{C}$-rational iff it is definable in $\mathcal{F}$ (as a fragment of MSOT).

We will obtain such a result, but with additional assumptions on $\mathbf{C}$ and $\mathcal{F}$. These assumptions are verified by the major classes for which such a correspondence holds, as for instance $\mathbf{C} = \mathbf{A}$ and $\mathcal{F} = \mathrm{FO}$. This will imply a decision procedure for deciding whether a rational function is definable in such fragments $\mathcal{F}$. All the fragments we consider include the total order $<$ over the domain, so we usually omit it, and write FO for FO[$<$] for instance.

$\mathcal{F}$-transducers. In Section 2.2.2, we defined MSO transducers (MSOTs) by interpreting output predicates over several copies of the input. These correspond exactly to regular functions,

[11] we name *fragment* of MSO any subset of formulas in MSO.

i.e. functions definable by two-way transducers [EH01]. In [Boj14, Fil15], is was shown that order-preserving MSOTs exactly capture rational functions, i.e. functions definable by one-way transducers. We give here a more direct definition of MSOTs that is equivalent to order-preserving MSOTs (introduced in Section 2.2.3), but more similar to one-way transducers. They operate on pointed words, that we define now.

A *pointed word* over an alphabet Σ is a pair (u, i) where $u \in \Sigma^+$ and $i \in \mathrm{dom}(u)$. Equivalently, we will see it as a logical structure over the signature $\{\mathsf{c}, (a(x))_{a\in\Sigma}, x < y\}$ where c is a constant symbol. MSO_c formulas (resp. $\mathcal{F}_\mathsf{c}$ formulas) are formulas obtained by taking an MSO formula (resp. $\mathcal{F}$ formula) and substituting some occurrences of first-order variables inside predicates by c. Given a MSO_c sentence ψ, we write $(u, i) \models \psi$ when u satisfies ψ using the usual semantics of MSO, and c is interpreted as i. The set of such pointed words (u, i) defines the *pointed language* $[\![\psi]\!]$ of ψ. Using pointed words avoids to dedicate a variable as a free variable, and this makes fragments with restrictions on variables more expressive, as for instance for $\mathcal{F} = \mathrm{FOT}^2$, the first-order fragment of MSOT where only two variables are allowed.

An *order-preserving MSO transducer* (1MSOT) over Σ, Δ is a tuple:

$$\mathcal{T} = (V, \phi_{\mathrm{dom}}, (\psi_v)_{v\in V})$$

where V is a finite set of words over Δ, ϕ_{dom} is an MSO sentence over Σ, and each ψ_v, for $v \in V$, is an MSO_c sentence. It defines a function (a relation in a more general case) in $\Sigma^* \to \Delta^*$ with domain $[\![\phi_{\mathrm{dom}}]\!]$ and such that: $[\![\mathcal{T}]\!](u) = v_1 \cdots v_{|u|}$ where $(u, i) \models \psi_{v_i}$, for $i \in \mathrm{dom}(u)$. Hence, ψ_v holds at pointed positions where v can be output. We will always assume that $[\![\mathcal{T}]\!]$ is indeed functional and well-defined. We also only consider functions that do not contain the empty word in their domain. As already mentioned in Section 2.2.3, a function is definable by an 1MSOT iff it is rational [Boj14, Fil15].

Given a fragment $\mathcal{F}$ of MSO, an *$\mathcal{F}$-transducer* over Σ is a 1MSOT $(V, \phi_{\mathrm{dom}}, (\psi_v)_{v\in V})$ where ϕ_{dom} is an $\mathcal{F}$-sentence, and ψ_v is an $\mathcal{F}_\mathsf{c}$-sentence for every $v \in V$. If f is realized by an $\mathcal{F}$-transducer, then we say that f is *$\mathcal{F}$-definable*. In particular we write 1FOT for the first-order fragment of 1MSOT.

C-transducers vs $\mathcal{F}$-transducers. Now that we have defined $\mathcal{F}$-transducers, i.e. our "logical" transducers, we would like to establish a link with "machine" transducers, i.e. **C**-transducers. Let us define four conditions on $\mathcal{F}$ that will be sufficient to prove this correspondence:

1. $\mathcal{F}_\mathsf{c}$-formulas over an alphabet Σ and $\mathcal{F}$-formulas over the extended alphabet $\Sigma \uplus \dot{\Sigma}$ define the same pointed languages.
2. A language over the alphabet Σ is definable by an $\mathcal{F}$-formula over Σ if and only if it is definable by an $\mathcal{F}$-formula over a larger alphabet $\Sigma \cup \Gamma$.
3. $\mathcal{F}$-languages are closed under *pointed concatenation*, meaning that for any two $\mathcal{F}$-languages L_1, L_2 over an alphabet Σ and a fresh symbol $\sharp$, $L_1 \cdot \sharp \cdot L_2$ is an $\mathcal{F}$-language over $\Sigma \uplus \{\sharp\}$.
4. $\{\epsilon\}$ is an $\mathcal{F}$-language.

Theorem 5.5. *If **C** is a congruence class equivalent to a fragment $\mathcal{F}$ verifying conditions (1)-(4), then:*

*a function is a **C**-rational iff it is $\mathcal{F}$-definable.*

The proof of this theorem uses an intermediate model using pairs of $\mathcal{F}$-formulas that are interpreted on pointed words, where the left (resp. right) formula is interpreted on the prefix

(resp. suffix) of the input preceding (resp. succeeding) the pointed position, with an additional test on the letter at the pointed position. This models is closer to bimachines, and thus it is easily shown to be equivalent to **C**-bimachines (without final outputs), which are themselves shown equivalent to **C**-transducers (provided that the class $\{\epsilon\}$ can be defined in **C** over any alphabet). Finally, it is shown that under conditions (1)-(4), pairs of $\mathcal{F}$-formulas and $\mathcal{F}$-transducers coincide in expressiveness.

It can be checked that conditions (1)-(4) are verified by the congruence classes **A**, **DA** and **J**, introduced in Section 5.1.1. Moreover, for each of these classes, a correspondence with a logical fragment has already been established:

- the first-order fragment FO of MSO consisting in MSO formulas where second-order variables are not allowed, and the class **A** of aperiodic congruences [Sch65, MP71],
- the fragment FO^2 of FO where only two variables are allowed, and the class of congruences **DA**,
- the set $\mathcal{B}\Sigma_1$ of formulas being a Boolean combination of existential FO formulas, and the class **J** of $\mathcal{J}$-trivial congruences.

Moreover all these congruence classes are decidable (their corresponding equations can be checked on the syntactic congruence of the considered language), so we can combine these decision procedures and Theorem 5.5 to show that:

Theorem 5.6. *Given a one-way transducer realizing a function f, the following three decision problems are decidable:*

- *Is f FO-definable?*
- *Is f FO^2-definable?*
- *Is f $\mathcal{B}\Sigma_1$-definable?*

We have seen that conditions (1)-(4) are sufficient, but they are probably not necessary. For instance, the logic FO[+1] does not satisfy condition (3), but there is good hope that FO[+1]-definability is decidable.

Open problem 6 (FO[+1]-*definability of one-way transducers*)

Determine if the following problem is decidable: given a one-way transducer realizing a function f, decide if f is FO[+1]*-definable.*

5.2 Rational functions over infinite words

The previous section presented a procedure to decide algebraic properties of rational functions, i.e. whether a rational function over finite words is **C**-rational, for a given congruence class **C** (as for instance $\mathbf{C} = \mathbf{A}$ in the aperiodic case). In turn, this provided a procedure for deciding whether a rational function is expressible in some logical fragment of MSO, whenever this fragment has a corresponding decidable congruence class. The present section aims at lifting these results to rational functions over *infinite* words, i.e. functions mapping infinite words to finite or infinite words.

Outline. We start by defining infinite words, and rational functions over such structures in Section 5.2.1. Then, in Section 5.2.2, we analyze two classes: sequential and quasi-sequential transductions. Bimachines are introduced in Section 5.2.3, and the canonical bimachine in Section 5.2.4, permitting to decide aperiodicity. The link with logics is then established in Section 5.2.5.

5.2.1 Infinite words and rational functions

Words and languages. An *infinite word* over a finite alphabet Σ is an infinite sequence of letters of Σ (or equivalently, a function from $\mathbb{N}$ to Σ). The set of infinite words over Σ is denoted Σ^ω, and we call $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ the set containing all finite and infinite words over Σ. Among infinite words, we will usually consider regular ones, i.e. those of the form uv^ω with $u, v \in \Sigma^*$ (called *ultimately periodic*). We define a normal form for these regular words: uv^ω is in *normal form* if v has minimal length and is minimal in the lexicographic order among all possible decompositions of uv^ω, and v is not a suffix of u (if $v \neq \epsilon$). Note that this normal form also applies to finite words, by taking $v = \epsilon$.

Most of the notions defined on finite words (in Section 2.1.1) are implicitly adapted to infinite words in the natural way, in particular the domain of a word, the prefix and suffix relations, etc. We prefer to focus on the differences. As a convention, we try to use the letters $u, v, w, \ldots$ for finite words (and also words in Σ^∞) and $x, y, z, \ldots$ for infinite words.

Given an infinite word $x \in \Sigma^\omega$, we denote by $\mathrm{Inf}(x)$ the set of letters of Σ appearing infinitely often in x. The closure $\overline{L}$ of $L \subseteq \Sigma^\infty$ is the set $\{u \in \Sigma^\infty \mid \forall i \in \mathbb{N},\ i \leq |u|,\ \exists w \text{ s.t. } u[1,i]w \in L\}$ i.e. the set of words such that any finite prefix has a continuation in L.

Automata. We use automata on infinite words with Muller acceptance conditions, and simply call them *automata* in the sequel. A *Muller automaton* over Σ is a tuple $\mathcal{A} = (Q, \delta, I, F)$ where Q is a finite set of *states*, $\delta \subseteq Q \times \Sigma \times Q$ is the set of *transitions*, $I \subseteq Q$ is the set of *initial* states, and $F \subseteq \mathcal{P}(Q)$ is the *final condition*.

A *run* of $\mathcal{A}$ over a word $w \in \Sigma^\infty$ is itself a word $r \in Q^\infty$ of length $|w| + 1$, (with the convention that $\infty + 1 = \infty$) such that for any $1 \leq i < |r|$, we have $(r[i], w[i], r[i+1]) \in \delta$. A run r is called *initial* if $r[1] \in I$, *final* if $r \in Q^\omega$ and $\mathrm{Inf}(r) \in F$, and *accepting* if it is both initial and final. A word is *accepted* by $\mathcal{A}$ if there exists an accepting run over it, and the language *recognized* by $\mathcal{A}$ is the set of words it accepts, denoted by $[\![\mathcal{A}]\!] \subseteq \Sigma^\omega$. A language $L \subseteq \Sigma^\omega$ is *ω-regular* if it is accepted by some automaton.

Like in the finite case, we will use automata to define bimachines, hence we need some notions of "left-to-right determinism" and "right-to-left determinism". An automaton is *deterministic* if it has a single initial state, and at most one initial run per word. A *left automaton* is a deterministic automaton with no final condition (that is, $F = \mathcal{P}(Q)$). A *right automaton* is an automaton for which every infinite word has exactly one final run. These were introduced in [CM03] under the name *prophetic automata*, where it is proved that any ω-regular language can be recognized by such a right automaton, while it is well-known that it can also be recognized by a deterministic automaton (with Muller conditions). We will always assume that automata (except right automata) are trim, without loss of generality.

Transducers. We define transducers on top of automata in the usual way: A *transducer* $\mathcal{T}$ over alphabets Σ, Δ is a tuple $(\mathcal{A}, \mathsf{out}_i, \mathsf{out})$ where $\mathcal{A}$ is an automaton over Σ, $\mathsf{out}_i : Q \to \Delta^*$ is the *initial function* and $\mathsf{out} : \delta \to \Delta^*$ is the *output function*. If r is an accepting run of $\mathcal{A}$ on an infinite word $x \in \Sigma^\omega$, and $v = \mathsf{out}(r[1], x[1], r[2])\mathsf{out}(r[2], x[2], r[3]) \cdots$ then we say that

$(x, \mathsf{out}_i(r[1])v)$ is realized by $\mathcal{T}$. Then $[\![\mathcal{T}]\!]$ denotes all such pairs realized by $\mathcal{T}$, and we name it the *transduction realized by* $\mathcal{T}$.

In the sequel we only consider *functional* transducers, so we just name them "transducers". Functionality can be decided [Gir86] in PTime [Pri02, BCPS03]. A transduction (on infinite words) is *rational* iff it can be realized by a transducer. A *deterministic* transducer is a transducer with a deterministic underlying automaton. A transduction is *sequential* if it is realized by a deterministic transducer.

Congruences. In our setting, right congruences will only operate on finite words. Left congruences are extended to infinite words with the same constraint: an equivalence relation $\approx$ over Σ^ω is a *left congruence* if, for all $x, y \in \Sigma^\omega$ and all $a \in \Sigma$, $x \approx y \to ax \approx ay$. We will only consider left congruences of finite index and such that each class is an ω-regular language. The definition of a congruence does not change: it is a left congruence and a right congruence, hence on finite words only.

Let us now define some congruences associated with automata. Given an automaton $\mathcal{A}$ with state space Q, the *right congruence associated with* $\mathcal{A}$ is defined for $u, v \in \Sigma^*$ by $u \sim_{\mathcal{A}} v$ if $\forall q \in Q$, there is an initial run of $\mathcal{A}$ over u reaching q if and only if there is one over v. Note that for a left automaton, there is a bijection (up to adding a sink state) between Q and the equivalence classes of $\mathcal{A}$. Similarly, the *left congruence associated with* $\mathcal{A}$ is defined for $x, y \in \Sigma^\omega$ by $x \approx_{\mathcal{A}} y$ if $\forall q \in Q$ there is a final run of $\mathcal{A}$ over x from q if and only if there is one over y. Given a right automaton there is a bijection between Q and the equivalence classes of $\approx_{\mathcal{A}}$. Finally, the *transition congruence of* $\mathcal{A}$ is defined for $u, v \in \Sigma^*$ by $u \equiv_{\mathcal{A}} v$ if $\forall p, q \in Q$, there is a run over u from p to q if and only if there is one over v. Like in the final case, an automaton is called *aperiodic* if its transition congruence is aperiodic. A language is called *aperiodic* if there exists an aperiodic automaton recognizing it. A transducer is *aperiodic* if its underlying automaton is aperiodic and in that case the transduction it realizes is called *aperiodic*.

5.2.2 Sequential and quasi-sequential transductions

In this section, some key algebraic objects are defined, notably the syntactic congruence of a function. Using this, we characterize sequential transductions. We then characterize quasi-sequential transductions, which are somehow sequential transductions on infinite words, allowed to add a final output "after" the computation.

$\widehat{f}$ and $\overline{f}$. From a function $f : \Sigma^\omega \to \Delta^\infty$, we define two new functions:

- $\widehat{f} : \Sigma^* \to \Delta^\infty$ is defined by: $\widehat{f}(u) = \bigwedge\{f(ux) \mid ux \in \mathrm{dom}(f)\}$. It is exactly the same as in the finite case, except that its range is now over both finite and infinite words: $\widehat{f}(u)$ outputs the longest prefix of the output once u has been read.
- $\overline{f} : \Sigma^\omega \to \Delta^\infty$ is defined by: $\overline{f}(x) = \lim_n \widehat{f}(x[1,n])$, for $x \in \overline{\mathrm{dom}(f)}$. Intuitively, $\overline{f}$ outputs what can be output by considering only prefixes of the input. For instance, like sequential transductions, it cannot detect if there is an infinite number of a's in the input. We call $\overline{f}$ the *sequential extension* of f. In particular, if f is sequential, then $\overline{f}$ extends f over $\overline{\mathrm{dom}(f)}$.

Example 5.4. *We present three rational functions that constitute our running examples.*

1. *f_{ab} maps each word over $\{a, b\}$ with a finite number of a's to the subsequence of ab-factors. For instance $f_{ab}(abbab^\omega) = abab$ and $f_{ab}(b^\omega) = \epsilon$, while $(abb)^\omega \notin \mathrm{dom}(f_{ab})$.*
   - *$\widehat{f_{ab}}$ just extracts the ab-factors, as for instance $f_{ab}(babbaba) = abab$.*

- *$\overline{f_{ab}}$ is defined over $\overline{\mathrm{dom}(f_{ab})} = \{a,b\}^\omega$ and $\overline{f_{ab}}((ba)^\omega) = \lim_n \widehat{f_{ab}}((ba)^n) = \lim_n (ab)^{n-1} = (ab)^\omega$.*

2. *$f_{\#a}$ maps a word x over $\{a,b\}$ to a^ω if x contains an infinite number of a's, and to b^ω otherwise.*

   - *reading a finite prefix u does not give any insight on the output, thus $\widehat{f_{\#a}}(u) = \epsilon$.*
   - *$\overline{f_{\#a}}(x) = \epsilon$ for every $x \in \{a,b\}^\omega$, as it is based on $\widehat{f_{\#a}}$.*

3. *f_{blocks} maps $u_1\# \dots \#u_n\#v$ where v does not contain $\#$, to $a_1^{|u_1|}\# \dots \#a_n^{|u_n|}\#w$ where $u_i \in \{a,b\}^*$, a_i is the last letter of u_i (if any), $w = a^\omega$ if v has an infinite number of a's, and $w = b^\omega$ otherwise.*

   - *$\widehat{f_{blocks}}(u_1\# \dots \#u_n\#v) = a_1^{|u_1|}\# \dots \#a_n^{|u_n|}\#$ whenever v does not contain $\#$ (for the same reason as for $f_{\#a}$),*
   - *$\overline{f_{blocks}}(u_1\# \dots \#u_n\#v) = a_1^{|u_1|}\# \dots \#a_n^{|u_n|}\#$ whenever v does not contain $\#$.*

Syntactic congruence. Given a transduction f, we define its *syntactic congruence* $\sim_f$ over Σ^* by $u \sim_f v$ if:

1. $\forall x \in \Sigma^\omega$, $ux \in \overline{\mathrm{dom}(f)} \Leftrightarrow vx \in \overline{\mathrm{dom}(f)}$, and

2. either $\widehat{f}(u)$ and $\widehat{f}(v)$ are both regular with the same period (in normal form), or they are both finite and $\forall x \in \Sigma^\omega$ such that $ux, vx \in \mathrm{dom}(f)$, $\widehat{f}(u)^{-1} f(ux) = \widehat{f}(v)^{-1} f(vx)$.

It can be checked that $\sim_f$ is a right congruence. Intuitively, $u \sim_f v$ means that (1) u and v, as prefixes, behaves identically w.r.t. $\overline{\mathrm{dom}(f)}$, and (2) $\widehat{f}(u)$ and $\widehat{f}(v)$ are infinite and identical up to a finite prefix, or they are finite and the remaining output for $f(ux)$ and $f(vx)$ is the same for all x, once $\widehat{f}(u)$ and $\widehat{f}(v)$ are removed. Hence it is very similar to the finite case, the main difference being the domain $\overline{\mathrm{dom}(f)}$.

Based on the syntactic congruence $\sim_f$, one can define a transducer $\mathcal{T}_f$, also in the same vein as in the finite case (with some additional technicalities). It can be proved that $\mathcal{T}_f$ realizes $\overline{f}$, and can be computed in PTime if f is given by a deterministic transducer.

Sequential transductions. We can now characterize sequential transductions:

Theorem 5.7. *A rational function f over infinite words is sequential iff:*

1. *$\sim_f$ has finite index, and*

2. *$\overline{f}_{|\mathrm{dom}(f)} = f$.*

We will see that condition (1) is equivalent to the weak twinning property in [BC04], hence this theorem adapts a result from [BC04] to transducers that can output finite words (not only infinite ones). Condition (2) states somehow that the output can be produced progressively, i.e. does not depend on an infinitary condition. When removing it, one obtains a new class of transductions with interesting properties: quasi-sequential transductions.

Quasi-sequential transductions. We name *quasi-sequential* transduction any rational transduction f which syntactic congruence $\sim_f$ has finite index. This is comparable with the notion of deterministic transducers on finite words, that can append a word to the output depending on the final state. Here, quasi-sequential transductions have a similar characterization by transducers allowed to append an output word in Δ^∞ according to the reached final condition (assuming the run produced a finite output so far). We do not elaborate on this "machine" view, but rather to the algebraic properties of quasi-sequential transductions.

We will see that quasi-sequential transductions are exactly those satisfying the weak twinning property defined in [BC04]. In order to define this notion, we need to identify constant states: a state is *constant* if all the final runs from this state produce the same word. Recall that $\mathsf{del}(u, v)$ measures the delay between words u and v, as defined in Section 4.2.4 page 72. A transducer $\mathcal{T}$ satisfies the *weak twinning property* if for any initial runs $p_1 \xrightarrow{u|\alpha_1} q_1 \xrightarrow{v|\beta_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2} q_2 \xrightarrow{v|\beta_2} q_2$ the following property holds:

- If q_1, q_2 are not constant then $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) = \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$.
- If q_1 is not constant, q_2 is constant and produces the regular word γ, then either $\beta_1 = \epsilon$ or $i(p_1)\alpha_1\beta_1^\omega = i(p_2)\alpha_2\beta_2\gamma$.

Note that if q_2 is constant and $\beta_2 \neq \epsilon$ then $\gamma = \beta_2^\omega$.

One key result in [BC04] is a determinization procedure for transducers over infinite words (that we call *subset construction with delays*) that terminates iff the transducer satisfies the weak twinning property. In fact one can show that, in this case, the resulting transducer realizes exactly $\overline{f}$. This will be used later to compute a canonical look-ahead. We can state the following characterization of quasi-sequential transductions:

Theorem 5.8. *If $\mathcal{T}$ is a transducer over infinite words, then the following statements are equivalent:*

1. *$\mathcal{T}$ satisfies the weak twinning property,*
2. *the transducer $\mathcal{S}$ obtained from $\mathcal{T}$ by the subset construction with delays is finite,*
3. *f is quasi-sequential.*

If these statements hold, then $\mathcal{S}$ is aperiodic when $\mathcal{T}$ is.

The equivalence between (1) and (2) is proved in [BC04].

5.2.3 Rational transductions

We have seen that sequential transductions can produce their output in a deterministic manner, and are not able to take infinitary conditions on the input into account. Quasi-sequential functions extend them a bit, by being able to produce their output in a non-progressive manner.

In this section, we consider the whole class of rational functions over infinite words. In order to have a deterministic device (on which algebraic properties can be studied), we generalize bimachines from finite to infinite words.

Bimachines. A *bimachine* on infinite words over alphabets Σ, Δ is similar to the finite case, but using a right automaton on infinite words implies some minor changes. Formally, a bimachine is a tuple $\mathcal{B} = (\mathcal{L}, \mathcal{R}, \mathsf{out}, \mathsf{out}_\mathsf{I})$ where $\mathcal{L} = (Q_\mathcal{L}, \delta_\mathcal{L}, \{l_0\})$ and $\mathcal{R} = (Q_\mathcal{R}, \delta_\mathcal{R}, I, F)$ are respectively a left and a right automaton, $\mathsf{out} : Q_\mathcal{L} \times \Sigma \times Q_\mathcal{R} \to \Delta^*$ is the *output function* and

$\mathsf{out}_\mathsf{l} : I \to \Delta^*$ is the *left final function*. We add the semantic restriction that $[\![\mathcal{L}]\!] = \overline{[\![\mathcal{R}]\!]}$ (a left automaton can only recognize closed languages). In order to define the output produced by $\mathcal{B}$ on an infinite word $u \in [\![\mathcal{R}]\!]$, let us define the word v_i produced when reading the position i of u. We have: $v_i = \mathsf{out}(l, u[i], r)$ where l is the unique state of $\mathcal{L}$ reached after reading $u[1, i-1]$ from the state l_0 (if defined), and r is the state of the unique final run of $\mathcal{R}$ on $u[i+1,]$ (this denotes the suffix of u starting at position $i+1$), if defined. Then the output produced by $\mathcal{B}$ on u is $\mathsf{out}_\mathsf{l}(r_0)v_1v_2\cdots$ where r_0 is the leftmost state of the unique final run of $\mathcal{R}$ on u (if defined). Hence bimachines on infinite words are defined in a similar way as in the finite case, except that the right automaton works a bit differently, and no right final condition is used. The transduction $[\![\mathcal{B}]\!]$ realized by $\mathcal{B}$ is defined over $[\![\mathcal{R}]\!]$.

Left minimization. In the finite case we defined the bimachine $\mathit{Left}(\mathcal{R})$ from a right automaton $\mathcal{R}$. This was based on the additional functions $\widehat{f}_{[w]_\mathcal{R}}$. We adapt these definitions quite naturally, and also the definition of $\overline{f}$ from the previous section:

- $\widehat{f}_x : \Sigma^* \to \Delta^\infty$ defined by $\widehat{f}_x(u) = \wedge\{f(uy) \mid y \approx_\mathcal{R} x\}$. Recall that $\approx_\mathcal{R}$ is the left congruence associated with $\mathcal{R}$.
- $\overline{f}^\mathcal{R} : \Sigma^\omega \to \Delta^\infty$ defined by $\overline{f}^\mathcal{R}(x) = \lim_n \widehat{f}_{x[n+1,]}(x[1,n])$.

Intuitively, $\widehat{f}_x(u)$ is the longest output that can be safely produced after reading the prefix u, provided that the suffix will be in the class of x, while $\overline{f}^\mathcal{R}(x)$ applies $\widehat{f}$ to prefixes of x progressively.

The next step, as in the finite case, is the definition of a right congruence from the definition of $\widehat{f}$. We call it the *$\mathcal{R}$-syntactic congruence of f* and define it over Σ^* by letting $u \sim_f^\mathcal{R} v$ if:

1. $\forall x \in \Sigma^\omega$, $ux \in \mathrm{dom}(f) \iff vx \in \mathrm{dom}(f)$, and
2. for any $x \in \Sigma^\omega$, either $\widehat{f}_x(u)$ and $\widehat{f}_x(v)$ are both infinite with the same ultimate period (in normal form) or they are both finite and $\widehat{f}_x(u)^{-1} f(ux) = \widehat{f}_x(v)^{-1} f(vx)$.

Now, from a right automaton $\mathcal{R}$, we define $\mathit{Left}(\mathcal{R})$ based on $\sim_f^\mathcal{R}$, and the bimachine $\mathcal{B}_f^\mathcal{R} = (\mathit{Left}(\mathcal{R}), \mathcal{R}, \mathsf{out}_\mathsf{l}^{f,\mathcal{R}}, \mathsf{out}^{f,\mathcal{R}})$ where $\mathsf{out}_\mathsf{l}^{f,\mathcal{R}}$ and $\mathsf{out}^{f,\mathcal{R}}$ output a maximal amount of information on finite parts, and one period per input letter otherwise:

- $\mathsf{out}^{f,\mathcal{R}}([u], a, [x]^\mathcal{R}) = \begin{cases} \widehat{f}_{ax}(u)^{-1}\widehat{f}_x(ua) & \text{if } \widehat{f}_x(ua) \text{ is finite} \\ \beta & \text{if } \widehat{f}_{ax}(u) = \alpha\beta^\omega,\ \beta \neq \epsilon \\ \alpha & \text{if } \widehat{f}_{ax}(u) \text{ is finite, } \widehat{f}_{ax}(u)^{-1}\widehat{f}(ua) = \alpha\beta^\omega \\ & \quad \text{and } \beta \neq \epsilon \end{cases}$
- $\mathsf{out}_\mathsf{l}^{f,\mathcal{R}}([x]^\mathcal{R}) = \begin{cases} \widehat{f}_x(\epsilon) & \text{if } \widehat{f}_x(\epsilon) \text{ is finite} \\ \alpha & \text{if } \widehat{f}_x(\epsilon) = \alpha\beta^\omega, \beta \neq \epsilon \end{cases}$

As in the sequential case, $\mathcal{B}_f^\mathcal{R}$ can be computed in PTime from a bimachine realizing f with right automaton $\mathcal{R}$. One can show that if a transducer with an underlying automaton $\mathcal{A}$ has a left congruence $\approx_\mathcal{A}$ coarser than $\approx_\mathcal{R}$ for some right automaton $\mathcal{R}$, then $\sim_\mathcal{A} \sqsubseteq \sim_f^\mathcal{R}$ and $\mathcal{B}_f^\mathcal{R}$ realizes f. This is used to transfer algebraic properties between transducers and bimachines. The proof also uses the fact that, from a left congruence, one can compute in 2ExpTime a right automaton recognizing it:

Theorem 5.9. *A function over infinite words is rational (resp. rational and aperiodic) iff it can be realized by a bimachine (resp. an aperiodic bimachine).*

5.2.4 Canonical bimachine

In this section we define a canonical bimachine for any rational function over infinite words. By canonical, we mean that two bimachines realizing the same function will have the same canonical bimachine. Our goal is not only to define such a machine, but also that this machine can be used to decide the algebraic properties we are interested in (here, aperiodicity).

We have seen in the previous section how to "left-minimize" a right automaton $\mathcal{R}$. Thus, the missing piece is to define a canonical right automaton. This one has to fulfill two constraints: being coarse enough to preserve algebraic properties, and being fine-grained enough to find a corresponding deterministic left automaton (hence a bimachine).

We proceed in two steps:

1. we define the *delay congruence* $\overset{\Delta}{\approx}_f$, the coarsest left congruence such that any right automaton $\mathcal{R}$ recognizing it satisfies that $f_{\mathcal{R}}$ is quasi-sequential. In other terms, this permits to have a deterministic transducer with a look-ahead recognizing the rational function. Hence this is not fine enough to define a bimachine.

2. we introduce the *ultimate congruence* $\overset{\cup}{\approx}_f$. This congruence, when used as look-ahead, transforms any "quasi-sequential transducer" into a deterministic one.

Hence, by taking the intersection of these two left congruences, we obtain a left congruence that is fine enough to transform any transducer into a deterministic one, when this left congruence is used as look-ahead. We will see that this left congruence is coarse enough, in that it is aperiodic when f is.

Delay congruence. We have already defined a notion of delay between a pair of (finite) words. (see Section 4.2.4 page 72). We generalize it to infinite words in the natural way, and extend it with respect to a function, by considering the delays between possible outputs. For two infinite words $x, y \in \Sigma^\omega$ and a transduction f, we define $\mathsf{del}_f(x, y) = \{\mathsf{del}(f(ux), f(uy)) \mid ux, uy \in \mathrm{dom}(f)\}$.

The *delay congruence* $\overset{\Delta}{\approx}_f$ of a function f is the left congruence obtained by setting $x \overset{\Delta}{\approx}_f y$ for $x, y \in \Sigma^\omega$ if (1) for all $u \in \Sigma^*, ux \in \mathrm{dom}(f)$ iff $vx \in \mathrm{dom}(f)$ and (2) $|\mathsf{del}_f(x, y)| < \infty$. This left congruence originates from [RS91, BLN12].

The delay congruence has some key properties:

1. when used as a look-ahead, it transforms any rational function into a quasi-sequential function,

2. if f is aperiodic, then so is $\overset{\Delta}{\approx}_f$,

3. if $\mathcal{A}$ (resp. $\mathcal{R}$) is the underlying automaton (resp. right automaton) of a transducer (resp. bimachine) realizing f, then $\approx_{\mathcal{A}}$ (resp. $\approx_{\mathcal{R}}$) is finer than $\overset{\Delta}{\approx}_f$.

Ultimate congruence. The *ultimate congruence* of a rational function f is defined, for $x, y \in \Sigma^\omega$ by taking $x \overset{\cup}{\approx}_f y$ whenever, for all $u \in \Sigma^*$:

- $ux \in \mathrm{dom}(f) \Leftrightarrow uy \in \mathrm{dom}(f)$
- if $ux \in \mathrm{dom}(f)$ then $\widehat{f}(u) = \overline{f}(ux) \iff \widehat{f}(u) = \overline{f}(uy)$. Moreover, if $\widehat{f}(u) = \overline{f}(ux)$ then $f(ux) = f(uy)$.

The equality $\widehat{f}(u) = \overline{f}(ux)$ expresses that no look-ahead on x would help outputting $f(ux)$ (in a deterministic way). The following properties of the ultimate congruence will be useful for our decision procedure:

- if f is quasi-sequential, then $\overset{\cup}{\approx}_f$ has finite index,
- if f is aperiodic, then so is $\overset{\cup}{\approx}_f$,
- $\overset{\cup}{\approx}_f$ can be computed in 2ExpTime from a bimachine realizing f.

Canonical bimachine. Let us now define how the delay congruence and the ultimate congruence are composed in order to define the canonical bimachine. This is basically an intersection (product construction), but where the ultimate congruence has access to the state (equivalence class) of the delay congruence.

Given two right automata $\mathcal{R}_1 = (Q_1, \delta_1, I_1, F_1)$ over Σ and $\mathcal{R}_2 = (Q_2, \delta_2, I_2, F_2)$ over $\Sigma \times Q_1$, the right automaton $\mathcal{R}_1 \bowtie \mathcal{R}_2$ is defined as $(Q_1 \times Q_2, \delta_{\{1,2\}}, I_1 \times I_2, F_1 \times F_2)$ with $F_1 \times F_2 = \{P_1 \times P_2 \mid P_1 \in F_1,\ P_2 \in F_2\}$ and $\delta_{\{1,2\}} = \{((s_1, s_2), a, (r_1, r_2) \mid (s_1, a, r_1) \in \delta_1,\ (s_2, (a, r_1), r_2) \in \delta_2\}$.

Given a left congruence $\approx$, we name *canonical automaton associated with* $\approx$ the right automaton obtained by (an adaptation of) the procedure described in [CM03], that builds a Büchi automaton recognizing a language from its syntactic congruence. We obtain the following result:

Theorem 5.10. *Let f be a rational function, $\mathcal{R}_1$ the canonical automaton associated with the delay congruence $\overset{\Delta}{\approx}_f$, and $\mathcal{R}_2$ the canonical automaton associated with the ultimate congruence $\overset{\cup}{\approx}_{f_{\mathcal{R}_1}}$. Then the bimachine $\mathcal{B}_f^{\mathcal{R}_1 \bowtie \mathcal{R}_2}$:*

- *realizes f, and*
- *is aperiodic if f is.*

As a consequence, given a bimachine realizing f, it is decidable whether f is aperiodic.

For this reason we name $\mathcal{B}_f^{\mathcal{R}_1 \bowtie \mathcal{R}_2}$ *the canonical bimachine associated with* f. This bimachine permits to decide the aperiodicity of the function, but cannot be used to decide the membership to any class of congruences, as we did in the finite case. In this sense it is "canonical for aperiodicity".

5.2.5 First-order definability

As we have seen, the algebraic notion of aperiodicity is decidable for rational functions over infinite words. As in the finite case, we will see that this transfers to logics.

First, let us remark that MSO logics for languages of infinite words can be defined in the same way as in the finite case, with some extra care, as for instance the fact that the output must belong to Δ^∞, while several copies of the input are available. This also holds for 1MSOT, i.e. order-preserving MSO transducers.

Despite these technicalities, the same techniques can be adapted, and we obtain the same correspondence between aperiodic rational functions and first-order definable order-preserving transductions (1FOT).

Proposition 5.6. *A rational transduction over infinite words is aperiodic iff it can be realized by a 1FOT transducer.*

Together with Theorem 5.10, this gives a procedure to decide whether a rational function is definable by a first-order transducer:

Theorem 5.11. *Given a bimachine (or a transducer) realizing a rational function f over infinite words, it is decidable whether f is realizable by a 1FOT transducer.*

Complexity. Throughout this chapter, complexity was not our main concern. Our decision procedures all rely on a fixed number of compositions of steps, and each of them can be performed in a fixed number of exponentials in time complexity. Thus, all the procedures presented in this chapter have an elementary time complexity.

Chapter 6

Conclusion and Perspectives

In this last chapter we take a final tour on the results presented in this manuscript, while modestly proposing some possible extensions and new directions.

6.1 Analyzing two-way transducers

We have seen in Chapter 3 and Section 4.1 how to analyze a two-way transducer, in order to answer the following questions:

- is it definable by a one-way transducer?
- if not, is it definable by a sweeping transducer? How many sweeps are needed in this case?
- equivalently, is it definable by a concatenation-free SST? How many registers are needed in this case?

Improving Rabin-Scott and Sherpherdson approaches. In Section 3.3 we presented the Rabin-Scott proof that two-way automata can be simulated by one-way automata. We have seen in Section 3.3 how this could be adapted to transducers, through the one-way definability problem. The major drawback of this proof lies in its time complexity: it is non-elementary.

Any attempt to recover elementary complexity would require to combine several steps of z-motion elimination in one step. This may be for instance obtained by removing several nested z-motions in one step, or by combining the guesses on the periods. At this point, we are tempted to conclude like Rabin and Scott did in their paper [RS59]: the Shepherdson approach seems an easier way.

Concerning the Shepherdson approach exposed in Section 3.4, the main open question here is to fill the complexity gap between the PSpace lower bound, and 2ExpSpace upper bound for deciding whether a two-way transducer is one-way definable. Our proof somehow builds a "best-effort" one-way transducer[1]. One may be tempted to look for a "direct criterion" on the two-way transducer, i.e. something like a twinning property on two-way runs. This is difficult to imagine so far, partly because one needs to quantify over all possible periods (in some range). Ismael Jecker proposed other new ideas that could yield an ExpSpace upper bound, but this work has not been published yet.

[1]The transducer $\mathcal{T}'$ described on page 56, is the "best under-approximation" of $\mathcal{T}$ among all transducers which domain can be pumped in the same way as $\mathcal{T}$. This has been formalized in [BGMP18, Corollary 8.7].

Output language. The expressive power of two-way transducers can be also considered on the output side, i.e. by considering the language of words that such a device can output. This output language may be not regular, and even not context-free. It is however k-iterative, as already mentioned in the introduction of Chapter 3: for every two-way transducer $\mathcal{T}$, there exists k and N in $\mathbb{N}$ such that every output word of $\mathcal{T}$ of length greater than N can be written as $u_1v_1u_2v_2\cdots u_kv_k$ (with $v_1v_2\cdots v_k \neq \epsilon$), and $u_0v_1^iu_1v_2^i\cdots u_{k-1}v_k^iu_k$ is also an output word of $\mathcal{T}$, for every i [Roz86, Smi14].[2] For instance, $k=1$ implies that the output language is regular, $k=2$ that it is context-free, etc. So it would be interesting to determine such a k:

Open problem 7 (*Degree of iterativity of two-way transducers' output languages*)

Is the following problem decidable: Given a two-way transducer (functional or not), determine the smallest k such that its output language is k-iterative.

As a side question, one may wonder how this smallest k increases when composing two two-way transducers? This would require a close inspection of the proof of closure by composition [CJ77]. Another variation would be to study the output of the iterated composition of two-way transducers (see for instance [BFH$^+$06] for sequential one-way transducers).

Determinism. The decision procedures we proposed were analyzing a non-deterministic (but functional) two-way transducer, and deciding whether it has an equivalent non-deterministic one-way transducer. We have seen in Proposition 4.2 that using bounded memory corresponds to being definable by a sequential transducer. Hence one may want to directly decide whether a two-way transducer has an equivalent *deterministic* one-way transducer (Open question 1). One can solve it by using the two-way to one-way procedure, followed by the sequentiality test [Cho77, BC02]. But one can hope for a better complexity. For instance, the resulting deterministic one-way transducer has to fulfill the twinning property, and this may add enough constraints to simplify our decision procedure.

One can go one step further by considering *multi-sequential* functions. These are functions that can be implemented by a finite union of deterministic transducers. Given a functional transducer, it is decidable whether it defines a multi-sequential function [CS86], in PTime [JF18]. This even holds when the transducer is not functional, i.e. defines a relation [JF18]. A "weak twinning property" permits to characterize such relations. This raises the question whether this can be extended to regular functions:

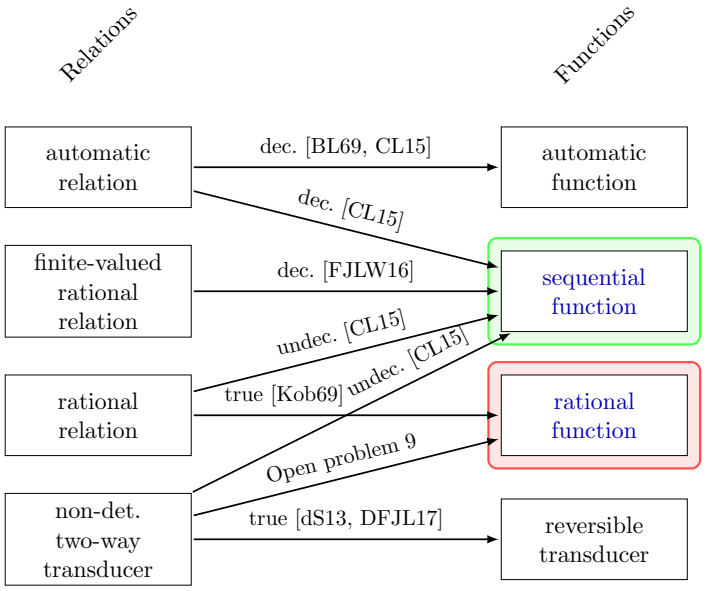
Open problem 8 (*Two-way to multi-sequential one-way transducers*)

Is the following problem decidable: Given a two-way transducer (possibly non-functional), is it equivalent to a multi-sequential (one-way) transducer?

This last question is of special interest, because multi-sequential relations are exactly those that can be evaluated with bounded memory [JF18].

Uniformization. This also relates to *uniformization*. Let us recall that a function f *uniformizes* a binary relation r if $\mathrm{dom}(f) = \mathrm{dom}(r)$ and $f \subseteq r$. As we have seen on page 25, one can always uniformize any two-way transducer (that may not be functional) by a reversible two-way transducer [dS13, DFJL17]. This is not true anymore when targeting deterministic transducers (i.e. sequential functions), because sequential uniformization is already undecidable for rational functions [CL15]. This is depicted in Figure 6.1. Note that this "sequential

[2] This is in contrast with, for instance, multiple context-free languages, which were proved *not* to be k-iterative for any k [KKM$^+$14].



An edge $R \xrightarrow{true} F$ means that, given a relation $r \in R$, it is always possible to uniformize it by a function $f \in F$. If not, dec. *and* undec. *edges indicate whether the corresponding decision problem is decidable or not.*

Figure 6.1: Overview of uniformization results.

uniformization" becomes decidable for finite-valued rational functions [FJLW16], and for automatic relations[3] [CL15]. Instead of targeting sequential functions, one may target rational functions. A one-way transducer can always be uniformized by a *functional* one-way transducer [Kob69]. This problem seems open when one allows a two-way transducer for defining the relation.

Open problem 9 (*Uniformization of two-way transducers by functional one-way*)

Can any two-way transducer (possibly not functional) be uniformized by a functional one-way transducer? If not, is the corresponding decision problem decidable?

Finite-valued transductions We mainly focused on functional transductions. A natural extension is to consider *finite-valued* relations, that is, binary relations for which there exists k such that each element of their domain has at most k images. Some interesting results have already been obtained.

For instance, one can minimize the number of registers of right-appending streaming string transducers being "finite-valued", in the specific sense that they remain deterministic, except the output function that becomes finite-valued (see [Rey15, page 84] and [DRT16]). This result is obtained thanks to a generalization of the twinning property.

Independently from the minimization of the number of registers, another ambitious challenge about finite-valued SSTs is to obtain a *decomposition theorem*, that is, to show that every finite-valued SST is equivalent to a union of functional transducers. This would imply, in particular, that finite-valued SSTs and two-way transducers have the same expressive power. In [GMPS17], such a decomposition is proven for finite-valued SSTs with one register. To my knowledge, it is

[3] A binary relation is *automatic* if it is recognized by an automaton reading a pair of input/output letters when firing a transition. A padding symbol is used when lengths differ.

still open when an arbitrary number of registers is allowed.

Open problem 10 (*Decomposition theorem for finite-valued SSTs*)

> *Is every finite-valued SST equivalent to a finite union of functional SSTs?*

The same problem is also open for finite-valued two-way transducers[4]. In fact, it is shown in [GMPS17] that, if the decomposition theorem holds for finite-valued SSTs, then it also does for finite-valued two-way transducers. Equivalence of finite-valued SSTs (resp. finite-valued two-way transducers) has been proved to be decidable by another technique [MP19a, CK86]. A decomposition theorem for one-way transducers has been established by Weber in [Web96] (see also [SdS10]).

In view of the main results of the present manuscript, a first step towards their generalization to finite-valued transducers would be:

Open problem 11 (*Finite-valued two-way to one-way transducers*)

> *Is the following problem decidable: Given a finite-valued two-way transducer, decide whether there exists an equivalent one-way transducer.*

Limited transducers Limited automata [Hib67, Pig19] constitute a slight extension of two-way automata. A d-limited automaton is a one-tape Turing machine where each cell is allowed to be rewritten only during its first d visits. 1-limited automata capture regular languages [WW86], while d-limited automata, with $d > 1$, capture context-free languages, even over a unary alphabet [PP13]. One could define transducers on this basis: A limited transducer would be a limited automaton with output words on its transitions, that would be concatenated progressively in a write-only one-way output tape. Clearly, 1-limited transducers can express all regular relations, as two-way transducers are 1-limited transducers. The converse seems less obvious.

Open problem 12 (1-*limited transducers*)

> *Do 1-limited transducers (resp. functional 1-limited transducers) exactly capture regular relations (resp. regular functions)?*

Polyregular functions In 2018, Bojańczyk introduced *polyregular functions* [Boj18], a class of word-to-word functions that strictly subsumes regular functions, and allows outputs of size polynomial in the size of the inputs (in contrast with the linear-size increase of regular functions). Polyregular functions enjoy several characterizations. One of them is two-way transducers, extended with pebbles having a stack discipline [EM02]. Interestingly, k-pebble transducers exactly define polyregular functions with output size in $O(n^k)$, and, for a given polyregular function, one can minimize the number of pebbles [Lho20]. In particular, for $k = 1$ one retrieves regular functions, hence it is decidable whether a polyregular function is regular. Other equivalent models are, among others, string-to-string MSO interpretations [BKL19], and regular list functions [BDK18].

Open problem 13 (*Polyrational functions*)

> *Define the class of* polyrational functions *as those definable by a one-way transducer with pebbles (with stack discipline). Has this class good algorithmic and closure properties? Does it enjoy equivalent characterizations, like a generalization of order-preserving, on MSO interpretations? Is the definability decidable, from polyregular to polyrational functions? This would generalize the one-way-definability procedures presented in Chapter 3.*

[4]The decidability of the finite-valuedness of two-way transducers has been recently proved [YY19].

For instance, the function that outputs all suffixes of a word is polyrational (and not regular), but the one that outputs all prefixes does not seem so. Closely related, a recent paper [DFG20] studies *marble transducers*, a restriction of two-way pebble transducers (with stack discipline) where pebbles (named marbles[5]) are only put to the left of the current position. This model is proved to be equivalent to copyful SSTs.

State complexity One of the most widely open problem concerning two-way automata is state complexity, usually referred as *minicomplexity*. In 1978, Sakoda and Sipser conjectured that there exist nondeterministic two-way automata for which equivalent deterministic two-way automata require an exponential number of states [SS78]. This conjecture has also consequences in terms of complexity classes (see e.g. [KP15]).

Still, some results have been obtained when restricting or extending the source or target class. For instance, the conjecture does not hold when the non-deterministic two-way automaton can make its non-deterministic choices only at the borders of the word (i.e. when reading the end-markers) [GGP14].

Restated on transducers, the Sakoda-Sipser conjecture becomes:

Open problem 14 (*State complexity of deterministic two-way transducers*)

> *Is there an exponential gap in the number of states, from functional two-way transducers, to deterministic two-way transducers?*

To my knowledge, the translations from functional two-way transducers to deterministic ones use crossing sequences and are thus exponential [Eng81] or the correspondence with MSOT [EH01], and no lower bound has been established. The constraint of yielding the output could help proving this conjecture on transducers (rather than automata) along.

Measuring two-wayness A question that naturally comes to mind when reasoning about two-way devices is: how many reversals are needed? For two-way automata, the answer is simply zero, because they are one-way definable. For two-way transducers, it is generally unbounded (for instance when reversing the input by blocks). We have described in Chapter 4 an algorithm deciding whether a functional two-way transducer can be implemented by an equivalent one with a uniformly bounded number of reversals (Corollary 4.2).

Beyond this bounded-vs-unbounded dichotomy, there may exist other measures to explore, like z-motion nesting degree. But a more direct measure is the number of registers of streaming string transducers.

Open problem 15 (*Register minimization of* SSTs)

> *Is the following problem decidable: Given a regular function f (given for instance as a two-way transducer, or an SST) and $k \geq 0$, can f be implemented by an SST with k registers?*

We have seen in Corollary 4.3 that this problem is decidable for concatenation-free NSSTs. As presented in the introduction of Chapter 4, this also holds for *right-appending* SSTs [DRT16, Rey15], and SSTs over a unary alphabet [AR13].

Two-way transducers over infinite words. In Section 5.2 we investigated the algebraic properties of *rational* functions over infinite words. These were defined by Muller transducers, that is, one-way transducers with Muller acceptance condition. Like in the finite case, a two-way extension of transducers gives raise to the class of *regular* functions over infinite words, with

[5] a notion introduced in [EHvB99] for trees.

equivalent models: two-way Muller transducers with regular look-ahead, MSOTs over infinite strings, and (functional, copyless) streaming string transducers [AFT12].

Recently, the decidability of the computability of regular functions over infinite words has been proved [DFKL19][6]. Here, computability means that the output can be progressively produced by a Turing machine on an output tape. This result reuses some properties of two-way loops exposed in Chapter 3, in particular idempotent loops. This naturally leads to the following question:

Open problem 16 (*Two-way to one-way transducers over infinite words*)

Is the following problem decidable: Given a regular function over infinite words (given by a two-way Muller transducer with regular look-ahead), is it rational (that is, is there an equivalent one-way Muller transducer)?

6.2 Pushdown and Trees

Two-way VPTs to one-way VPTs Another way to generalize two-way transducers is to equip them with a stack [GHI67]. In Chapter 4, we studied visibly pushdown transducers (VPTs), that is, one-way transducers equipped with a visible stack: each input letter indicates the operation (push/pop) on the stack. In particular we focused on the amount of memory required for evaluating the associated transduction. One step further is to consider *two-way* VPTs.

Open problem 17 (*Two-way to one-way* VPTs)

Is the following problem decidable: Given a functional two-way VPT, is it definable by a one-way VPT?

This problem is a generalization of the one-way definability of two-way transducers detailed in Chapter 3. Adding a stack breaks some parts of the proofs described in this manuscript. For instance two-way VPTs cannot be normalized in order to assume bounded crossing.

Numerous properties are already known about two-way VPTs [DFRT16, DFT19], including an equivalence with MSOT on nested words (when adding a single use restriction), and with copyless VPAs with registers (*à la* SSTs) [AD17]. Note that at the automaton level, two-way VPAs and one-way VPAs are known to be equi-expressive [MV09, DFRT16].[7]

VPTs to two-way transducers In terms of streamability, we could adapt our definition, and say that streamability corresponds to being "two-way sequential": the memory remains bounded, but we are allowed to read the input several times, in a deterministic manner. For words, every functional two-way transducer can be made deterministic [EH01] and even reversible [DFJL17], so this new streamability reduces to being definable by a functional two-way transducer. The question, for one-way VPTs, becomes:

> Is the following problem decidable: Given a VPT (functional or not), is it definable by a two-way transducer?

In fact the question is easily solved: In order to be definable by a two-way transducer, a VPT must have a regular domain (as two-way transducers do). And every VPT with a regular domain

[6] and extended to the case of data values (infinite alphabet) in [EFR20].

[7] In [BG17], a double-exponential lower bound is established, to convert two-way pushdown automata of constant height to one-way pushdown automata of constant height. However it does not apply to VPAs, as the constant-height restriction bounds their expressiveness to regular languages.

is rational, i.e. has an equivalent one-way transducer (because its domain can only contain words of bounded height, similarly to Proposition 4.4). And this is of course also sufficient for being definable by a two-way transducer. So a VPT is definable by two-way transducer iff it is rational, iff its domain is regular. The latter is decidable in PTime [Srb09].

This is somehow counter-intuitive, because two-way transducers can output languages that are context-free and not regular. For instance they can output the Dyck language of well-nested words with one type of parentheses [Roz86]. The point is that they do not produce it in a comparable manner: VPTs heavily rely on their domain, but not two-way transducers (when we consider the output language).

The situation is exactly the same when we start from a *two-way* VPT, because the regularity of the domain imposes that the height of the words in the domain is bounded.

Two-way vs one-way on trees. Visibly pushdown automata (resp. transducers) can also be used as tree automata (resp. transducers), by processing the depth-first linearization of the tree (resp. potentially also outputting the linearization of a tree [RT16][8]). The "historical" models of tree automata are different, as they operate from the leaves to the root (for bottom-up tree automata), or from the root to the leaves (top-down tree automata) [CDG+07].

Tree-walking automata constitute the "two-way" version of tree automata. When at a given node of a tree, a tree-walking automaton is allowed to move to its parent, or to one of its children, as indicated by a transition rule. Surprisingly, this "two-way" extension is less expressive than the "one-way" version: some regular tree languages are accepted by no tree-walking automata [BC08], because a tree-walking automaton can get "lost" in the tree. When extended with look-around, and restricted by a single-use discipline, tree-walking *transducers* recover the exact expressive power of tree-to-tree MSO [BE00, CE12] (another method uses pebbles instead [EHS07]).

However, on trees, several incomparable "one-way" transducer models compete. Indeed, top-down and bottom-up tree transducers are incomparable [Eng75]. Let us instantiate the one-way definability problem on top-down tree transducers:

Open problem 18 (*Tree-walking to top-down tree transducers*)

Is the following problem decidable: Given a single-use tree-walking transducer with look-around, does it have an equivalent top-down tree transducer?

Interestingly, memory requirements for transformations defined by tree-walking transducers started to be studied. In [EIM19], it is shown that by composing them, one can always keep the space usage linear in the output (and input) size, for instance. Definability and uniformization questions have also been considered on tree transducers, see e.g. [LW17, LS19].

On trees, other finite-state devices as expressive as tree-to-tree MSO transductions exist. Let us mention macro tree transducers with regular look-ahead [EM99], and also streaming tree transducers [AD17]. Both are "one-way" models, and thus can be considered as generalizations of SSTs on words, even though tree transformations are much more complex to analyze than word transformations. Consequently, many questions on word transformations can be adapted, as for instance the minimization of parameters of macro tree transducers, resp. of registers of streaming tree transducers.

[8] see also [MS18] for the case of top-down tree-to-string transducers.

6.3 Algebra and logics

In Chapter 5, we have considered the "algebraic" side of transductions, and the correspondence with some logics. In particular we have described an algorithm for deciding whether a rational function belongs to a given decidable congruence class (as for instance aperiodic congruences), and to an associated logic (for instance FOT). Then, we proved that the decidability also holds for transductions over infinite words, for the special case of aperiodic congruences and FOT.

Algebraic characterization of regular functions As already stated in Open Problem 4, the next challenge is to obtain such characterizations for regular functions. This problem is wide open. In particular, we lack a deterministic device like bimachines for two-way transducers (when considering the problem on machines). A first step could be to obtain this characterization for functions only performing a "back-and-forth" sweep on the input, and then try to move to sweeping transducers (which is, as we have seen, the same class as bounded-reversal transducers: both properties could help at the algebraic level).

A series of recent papers considered *reversible* automata and transducers [LPP17a, LPP17b, DFJL17, GKMP18]. Reversible means that the device is both deterministic and co-deterministic (that is, leftwards deterministic). Given the nature of bimachines, and their usage in our proofs, one could think of studying transductions definable by *reversible* one-way transducers, which form a strict subclass of rational transductions, and hope for a simpler proof of characterization. However, even at the level of languages, there is no unique minimal reversible automaton, given a regular reversible language [LPP17a][9]. And for regular functions, it makes no sense, because they are all reversible [DFJL17].

Some notions of semigroups or monoids have been defined for two-way automata [Bir89, Bir90, MSTV06, CD15] and for streaming string transducers [FKT14, DJR18], but they still do not lead to the definition of a deterministic device like bimachines. Beyond this "transducers" view (and their congruences), one may look for an alternative way to define word functions, more suitable to algebraic characterizations. One can think for instance of *monoid programs* [Bar89, GMS17], which both subsume monoid morphisms in terms of language recognition, and are less machine-oriented.

A recent trend is to use Hilbert's basis theorem in order to prove the decidability of equivalence (or functionality) for some classes of transducers [BDSW17, SMK18, BPS18, Boj19], or Ehrenfeucht's conjecture [MP19a]. Whether this method could help solve the aforementioned decidability problems is also to be investigated.

The algebraic approach also applies to richer structures, like trees [ÉW10, FSM11], but the situation is even more complex in this case.

Separation and covering of transductions First-order definability can be classified as a *membership* problem: is a function of class $\mathcal{C}$ also in a given class $\mathcal{C}'$? Such membership problems also apply on languages (rather than functions).

One of their generalizations is the *separation* problem. Consider two languages A, B from a class $\mathcal{C}$ (for instance, regular languages). A language S from a class $\mathcal{C}'$ (for instance, star-free languages) *separates* A, B if it contains A and does not intersect B. S is called a separator of A, B. The separation of $\mathcal{C}$ by $\mathcal{C}'$ is said decidable if there exists a procedure that takes A, B from $\mathcal{C}$ as inputs, and decides whether there exists a separator of A, B in $\mathcal{C}'$. This is the case in our example: a procedure deciding the separation of regular languages by star-free languages is described in [PZ14].

[9] but one can build a reversible automaton from a minimal one [Lom02].

Separation has been generalized to covering problems [PZ18]. Recently, separation and covering have been successfully used, for instance, for deciding membership of a regular language in some levels of the quantifier alternation hierarchy in first-order logic [PZ19]. Indeed, deciding separation requires a deep understanding of the expressive power of the separator class. This sometimes leads to undecidability results, as for instance the undecidability of the separation of visibly pushdown languages by regular languages [Kop16], and of the separation of regular tree languages by deterministic tree-walking automata [Boj17].

It seems that separation could be defined and studied on word-to-word functions too: a separator f_S from a word-to-word function class $\mathcal{C}'$ separates two word-to-word functions f_A, f_B of a class $\mathcal{C}$ if $f_A \subseteq f_S$ and $f_S \cap f_B = \emptyset$, when functions are described as input/output pairs. This implies in particular that $\mathrm{dom}(S)$ separates $\mathrm{dom}(A), \mathrm{dom}(B)$. So, if the separation of $\mathcal{C}$ by $\mathcal{C}'$ is decidable, so is the separation of the corresponding domain classes. But the converse is probably false: the domains could be separable, but not the functions, because this adds additional constraints to the transition system.

The separation of regular functions by rational functions is obvious. Given two regular functions f_A and f_B, if f_A is one-way definable then f_A and f_B are separable by a rational function iff $f_A \cap f_B = \emptyset$, which is decidable (in this case, f_A is a separator). Otherwise, if f_A is not one-way definable, then no function containing it is, so f_A and f_B are not separable by a rational function. As one-way definability is decidable (Theorem 2.2), the separation of regular functions by rational functions also is.

It seems less obvious to separate rational functions by first-order definable functions.

Open problem 19 (*Separation of functions*)

Is the separation of rational functions by FOT functions decidable?

The separation of *regular* functions by first-order definable functions is a generalization of Open problem 4. A further step would be to consider covering problems instead of separation problems.

To conclude...

Of course these perspectives are not exhaustive and reflect a personal point of view. For instance, alternative ways of defining transductions, as listed on page 17, also yield a number of open problems. A quantitative approach to transductions also conveys new problems and techniques [Ans90, Lom16, DL19, DG19, LMT19], as well as the setting of infinite alphabets (data values) [BS20].

Also, we followed a quite "theoretical" line, while sequential transducers are algorithms. They appeared as a way to abstract compilers, and for natural language processing. Hence, other lines of research follow a more "practical" view on transducers, and may deserve a joint effort with the "algorithms" community, for instance. Another way to move towards practical applications would be a certified implementation of our procedures (and more generally, of standard results on transducers), by means of a proof assistant like Coq [Coq, DS18].

Open problems

Index

Bibliography

[AC10] Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, pages 1–12, 2010. (Cited pages 11, 16, 30, and 31)

[AD11] Rajeev Alur and Jyotirmoy V. Deshmukh. Nondeterministic streaming string transducers. In *Automata, Languages and Programming*, pages 1–20. Springer Berlin Heidelberg, 2011. (Cited pages 16 and 32)

[AD17] Rajeev Alur and Loris D'Antoni. Streaming tree transducers. *Journal of the ACM*, 64(5):31:1–31:55, 2017. (Cited pages 18, 61, 106, and 107)

[ADD+13] Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013*, pages 13–22. IEEE Computer Society, 2013. (Cited page 18)

[AFM+20] Rajeev Alur, Dana Fisman, Konstantinos Mamouras, Mukund Raghothaman, and Caleb Stanford. Streamable regular transductions. *Theoretical Computer Science*, 807:15–41, 2020. (Cited page 18)

[AFR14] Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14*, pages 9:1–9:10. ACM, 2014. (Cited page 18)

[AFT12] Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS*, pages 65–74, 2012. (Cited pages 18, 33, and 106)

[AHU69] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. A general theory of translation. *Mathematical systems theory*, 3(3):193–221, Sep 1969. (Cited pages 9 and 16)

[AKL10] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Trans. Algorithms*, 6(2):28:1–28:36, 2010. (Cited page 61)

[AKMV05] Rajeev Alur, Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Congruences for visibly pushdown languages. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 1102–1114. Springer, 2005. (Cited page 61)

[Alu] Rajeev Alur. Nested words page: `https://www.cis.upenn.edu/~alur/nw.html`. (Cited page 61)

[AM09] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):16:1–16:43, 2009. (Cited page 61)

[AMS17] Rajeev Alur, Konstantinos Mamouras, and Caleb Stanford. Automata-Based Stream Processing. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 112:1–112:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. (Cited page 61)

[Ans90] Marcella Anselmo. Two-way automata with multiplicity. In *Automata, Languages and Programming, 17th International Colloquium, ICALP'90, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 88–102. Springer, 1990. (Cited page 109)

[AR13] Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'13, pages 37–48. Springer-Verlag, 2013. (Cited pages 60 and 105)

[Arn85] André Arnold. A syntactic congruence for rational omega-language. *Theoretical Computer Science*, 39:333–335, 1985. (Cited page 79)

[AU70] A.V. Aho and J.D. Ullman. A characterization of two-way deterministic classes of languages. *Journal of Computer and System Sciences*, 4(6):523 – 538, 1970. (Cited page 16)

[Bar71] Bruce H. Barnes. A two-way automaton with fewer states than any equivalent one-way automaton. *IEEE Transactions on Computers*, 20(4):474–475, 1971. (Cited page 36)

[Bar89] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989. (Cited page 108)

[Bas17] Félix Baschenis. *Minimizing resources for regular word transductions*. PhD thesis, Université de Bordeaux, 2017. (Cited pages 24 and 29)

[BC02] Marie-Pierre Béal and Olivier Carton. Determinization of transducers over finite and infinite words. *Theoretical Computer Science*, 289(1):225–251, 2002. (Cited pages 11, 16, 25, 26, 70, 73, 78, 83, and 102)

[BC04] Marie-Pierre Béal and Olivier Carton. Determinization of transducers over infinite words: The general case. *Theory of Computing Systems*, 37(4):483–502, 2004. (Cited pages 79, 95, and 96)

[BC08] Mikołaj Bojańczyk and Thomas Colcombet. Tree-walking automata do not recognize all regular languages. *SIAM Journal on Computing*, 38(2):658–701, 2008. (Cited page 107)

[BCF+07] Scott Boag, Don Chamberlin, Mary F. Fernàndez, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML query language, W3C recommendation, 2007. (Cited page 60)

[BCPS03] Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1):45 – 63, 2003. Selected Papers in honor of Jean Berstel. (Cited pages 70, 73, and 94)

[BDGP17] Mikołaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. Which Classes of Origin Graphs Are Generated by Transducers. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 114:1–114:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. (Cited pages 19 and 30)

[BDK18] Mikołaj Bojańczyk, Laure Daviaud, and Shankara Narayanan Krishna. Regular and first-order list functions. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 125–134. ACM, 2018. (Cited page 104)

[BDSW17] Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017. (Cited page 108)

[BE00] Roderick Bloem and Joost Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61(1):1–50, 2000. (Cited page 107)

[Ber79] Jean Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher : Informatik*. Teubner, 1979. (Cited pages 9, 10, 18, 78, and 80)

[BFH+06] Henning Bordihn, Henning Fernau, Markus Holzer, Vincenzo Manca, and Carlos Martín-Vide. Iterated sequential transducers as language generating devices. *Theoretical Computer Science*, 369(1-3):67–81, 2006. (Cited page 102)

[BG17] Zuzana Bednárová and Viliam Geffert. Two double-exponential gaps for automata with a limited pushdown. *Information and Computation*, 253:381–398, 2017. (Cited page 106)

[BGMP15] Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. One-way definability of sweeping transducer. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015*, volume 45 of *LIPIcs*, pages 178–191. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. (Cited pages 12, 23, and 24)

[BGMP16] Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Minimizing resources of sweeping and streaming string transducers. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. Full version available at `https://hal.archives-ouvertes.fr/hal-01274992`. (Cited pages 12 and 24)

[BGMP17] Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Untwisting two-way transducers in elementary time. In *32nd Annual ACM/IEEE Symposium*

on Logic in Computer Science, LICS 2017, pages 1–12, 2017. (Cited pages 12 and 24)

[BGMP18] Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. One-way definability of two-way word transducers. *Logical Methods in Computer Science*, Volume 14, Issue 4, 2018. (Cited pages 12, 23, 24, 36, 44, and 101)

[BHPS61] Yehoshua Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172, 1961. Reprinted in Y. Bar-Hillel. (1964). *Language and Information: Selected Essays on their Theory and Application*, Addison-Wesley 1964, 116–150. (Cited page 70)

[Bir89] Jean-Camille Birget. Concatenation of inputs in a two-way automaton. *Theoretical Computer Science*, 63(2):141–156, 1989. (Cited page 108)

[Bir90] Jean-Camille Birget. Two-way automaton computations. *Informatique Théorique et Applications*, 24:47–66, 1990. (Cited page 108)

[Bir93] Jean-Camille Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical Systems Theory*, 26(3):237–269, 1993. (Cited pages 36 and 38)

[BKL19] Mikołaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-string interpretations with polynomial-size output. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 106:1–106:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited pages 10 and 104)

[BKM+19] Sougata Bose, Shankara Narayanan Krishna, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. On synthesis of resynchronizers for transducers. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, volume 138 of *LIPIcs*, pages 69:1–69:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited page 19)

[BL69] J. Richard Büchi and Lawrence H. Landweber. Definability in the monadic second-order theory of successor. *Journal of Symbolic Logic*, 34(2):166–170, 1969. (Cited page 103)

[BLN12] Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning rational functions. In *Proceedings of the 16th International Conference on Developments in Language Theory (DLT)*, pages 273–283, 2012. (Cited page 98)

[BMPP18] Sougata Bose, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. Origin-equivalence of two-way word transducers is in PSPACE. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018*, volume 122 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. (Cited page 19)

[Boj14] Mikołaj Bojańczyk. Transducers with origin information. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2014. (Cited pages 9, 18, 29, and 91)

[Boj17] Mikołaj Bojańczyk. It is undecidable if two regular tree languages can be separated by a deterministic tree-walking automaton. *Fundamenta Informaticae*, 154(1-4):37–46, 2017. (Cited page 109)

[Boj18] Mikołaj Bojańczyk. Polyregular functions. *CoRR*, abs/1810.08760, 2018. (Cited pages 10 and 104)

[Boj19] Mikołaj Bojańczyk. The Hilbert method for transducer equivalence. *SIGLOG News*, 6(1):5–17, 2019. (Cited page 108)

[BPS18] Adrien Boiret, Radoslaw Piórkowski, and Janusz Schmude. Reducing transducer equivalence to register automata problems solved by "Hilbert method". In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018*, volume 122 of *LIPIcs*, pages 48:1–48:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. (Cited page 108)

[BR18] Nicolas Baudru and Pierre-Alain Reynier. From two-way transducers to regular function expressions. In *Developments in Language Theory*, pages 96–108. Springer International Publishing, 2018. (Cited pages 17 and 18)

[BS20] Mikołaj Bojańczyk and Rafał Stefański. Single-use automata and transducers for infinite alphabets. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited page 109)

[Büc60] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschr. f. math. Logik und Grundlagen d. Math.*, 6:66–92, 1960. (Cited pages 9, 15, 27, and 77)

[Car10] Olivier Carton. Right-sequential functions on infinite words. In *Computer Science - Theory and Applications, 5th International Computer Science Symposium in Russia, CSR 2010*, volume 6072 of *Lecture Notes in Computer Science*, pages 96–106. Springer, 2010. (Cited page 79)

[Car12] Olivier Carton. Two-way transducers with a two-way output tape. In *Developments in Language Theory*, pages 263–272, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cited page 18)

[CCP17] Michaël Cadilhac, Olivier Carton, and Charles Paperman. Continuity and rational functions. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 115:1–115:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. (Cited page 79)

[CD15] Olivier Carton and Luc Dartois. Aperiodic Two-way Transducers and FO-Transductions. In *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 160–174. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015. (Cited pages 80 and 108)

[CDG+07] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available on: `http://tata.gforge.inria.fr`, 2007. release October, 12th 2007. (Cited page 107)

[CE12] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, 1st edition, 2012. (Cited pages 9, 10, 16, 28, and 107)

[CES17] Olivier Carton, Léo Exibard, and Olivier Serre. Two-way two-tape automata. In *Developments in Language Theory*, pages 147–159. Springer International Publishing, 2017. (Cited page 18)

[CG14] Christian Choffrut and Bruno Guillon. An algebraic characterization of unary two-way transducers. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2014. (Cited page 80)

[CH91] Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is PSpace-complete. *Theoretical Computer Science*, 88(1):99–116, 1991. (Cited page 90)

[Cho77] Christian Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theoretical Computer Science*, 5(3):325–337, 1977. (Cited pages 9, 11, 16, 25, 78, and 102)

[Cho79] Christian Choffrut. A generalization of ginsburg and rose's characterization of G-S-M mappings. In *Automata, Languages and Programming, 6th Colloquium, ICALP'79, Proceedings*, volume 71 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 1979. (Cited page 78)

[Cho03] Christian Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131–143, 2003. (Cited pages 11, 12, 78, 82, and 84)

[CJ77] Michal Chytil and Vojtech Jákl. Serial composition of 2-way finite-state transducers and simple programs on strings. In *Proceedings of the Fourth Colloquium on Automata, Languages and Programming*, pages 135–147. Springer-Verlag, 1977. (Cited pages 22 and 102)

[CK86] Karel II Culik and Juhani Karhumäki. The equivalence of finite valued transducers (on HDT0L languages) is decidable. *Theoretical Computer Science*, 47(3):71–84, 1986. (Cited pages 23, 25, and 104)

[CK87] Karel II Culik and Juhani Karhumäki. The equivalence problem for single-valued two-way transducers (on npdtol languages) is decidable. *SIAM Journal on Computing*, 16(2):221–230, 1987. (Cited page 16)

[CKLP15] Michaël Cadilhac, Andreas Krebs, Michael Ludwig, and Charles Paperman. A circuit complexity approach to transductions. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015*, volume 9234 of *Lecture Notes in Computer Science*, pages 141–153. Springer, 2015. (Cited page 79)

[CL15] Arnaud Carayol and Christof Löding. Uniformization in automata theory. In *Logic, Methodology and Philosophy of Science - Proceedings of the 14th International Congress*, 2015. (Cited pages 102 and 103)

[Cla99] James Clark. XSL Transformations (XSLT) version 1.0, W3C recommendation, 1999. (Cited page 60)

[CM03] Olivier Carton and Max Michel. Unambiguous büchi automata. *Theoretical Computer Science*, 297(1-3):37–81, 2003. (Cited pages 79, 93, and 99)

[Col07] Thomas Colcombet. Factorisation forests for infinite words. In *Fundamentals of Computation Theory (FCT)*, volume 4639 of *LNCS*, pages 226–237. Springer, 2007. (Cited pages 54, 56, and 66)

[Coq] The Coq proof assistant. https://coq.inria.fr/. (Cited page 109)

[Cou94] Bruno Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53 – 75, 1994. (Cited pages 16 and 28)

[CRT15] Mathieu Caralp, Pierre-Alain Reynier, and Jean-Marc Talbot. Trimming visibly pushdown automata. *Theoretical Computer Science*, 578(C):13–29, 2015. (Cited pages 69 and 71)

[CS86] Christian Choffrut and Marcel Paul Schützenberger. Décomposition de fonctions rationnelles. In *STACS 86, 3rd Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 210 of *Lecture Notes in Computer Science*, pages 213–226. Springer, 1986. (Cited page 102)

[DDO19] Egor Dobronravov, Nikita Dobronravov, and Alexander Okhotin. On the length of shortest strings accepted by two-way finite automata. In *Developments in Language Theory - 23rd International Conference, DLT 2019, Proceedings*, volume 11647 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 2019. (Cited page 37)

[DFF19] María Emilia Descotte, Diego Figueira, and Santiago Figueira. Closure properties of synchronized relations. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPIcs*, pages 22:1–22:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited page 19)

[DFG20] Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. Register transducers are marble transducers. *CoRR*, abs/2005.01342, 2020. (Cited page 105)

[DFJL17] Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote. On Reversible Transducers. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 113:1–113:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. (Cited pages 10, 16, 25, 32, 60, 102, 103, 106, and 108)

[DFKL19] Vrunda Dave, Emmanuel Filiot, Shankara Narayanan Krishna, and Nathan Lhote. Deciding the computability of regular functions over infinite words. *CoRR*, abs/1906.04199, 2019. (Cited page 106)

[DFL18] Luc Dartois, Emmanuel Filiot, and Nathan Lhote. Logics for word transductions with synthesis. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 295–304. ACM, 2018. (Cited page 19)

[DFP18] María Emilia Descotte, Diego Figueira, and Gabriele Puppis. Resynchronizing classes of word relations. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 123:1–123:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. (Cited page 19)

[DFRT16] Luc Dartois, Emmanuel Filiot, Pierre-Alain Reynier, and Jean-Marc Talbot. Two-way visibly pushdown automata and transducers. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16*, pages 217–226. ACM, 2016. (Cited pages 61 and 106)

[DFT19] Luc Dartois, Emmanuel Filiot, and Jean-Marc Talbot. Two-way Parikh automata with a visibly pushdown stack. In *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019*, volume 11425 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2019. (Cited page 106)

[DG08] Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008. (Cited page 77)

[DG19] Manfred Droste and Paul Gastin. Aperiodic weighted automata and weighted first-order logic. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, volume 138 of *LIPIcs*, pages 76:1–76:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited page 109)

[DGK08] Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words. *International Journal of Foundations of Computer Science*, 19(3):513–548, 2008. (Cited page 77)

[DGK18] Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. Regular transducer expressions for regular transformations. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 315–324. Association for Computing Machinery, 2018. (Cited page 18)

[DJR18] Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier. Aperiodic string transducers. *International Journal of Foundations of Computer Science*, 29(5), 2018. (Cited pages 31, 32, 33, 60, 80, and 108)

[DKT16] Vrunda Dave, Shankara Narayanan Krishna, and Ashutosh Trivedi. FO-definable transformations of infinite strings. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, volume 65 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. (Cited page 79)

[DL19] Louis-Marie Dando and Sylvain Lombardy. From Hadamard expressions to weighted rotating automata and back. *Theoretical Computer Science*, 787:28–44, 2019. (Cited page 109)

[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*, pages 857–866. ACM, 2016. (Cited pages 60, 103, and 105)

[dS13] Rodrigo de Souza. Uniformisation of two-way transducers. In *Language and Automata Theory and Applications*, pages 547–558. Springer Berlin Heidelberg, 2013. (Cited pages 25, 102, and 103)

[DS18] Christian Doczkal and Gert Smolka. Regular language representations in the constructive type theory of coq. *Journal of Automated Reasoning*, 61(1-4):521–553, 2018. (Cited pages 38 and 109)

[EFR20] Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. On computability of data word functions defined by transducers. In *Foundations of Software Science and*

Computation Structures - 23rd International Conference, FOSSACS 2020, Proceedings, volume 12077 of *Lecture Notes in Computer Science*, pages 217–236. Springer, 2020. (Cited page 106)

[EH01] Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254, 2001. (Cited pages 9, 16, 25, 28, 29, 30, 31, 36, 91, 105, and 106)

[EH07] Joost Engelfriet and Hendrik Jan Hoogeboom. Automata with nested pebbles capture first-order logic with transitive closure. *Logical Methods in Computer Science*, 3(2), 2007. (Cited page 77)

[EHS07] Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML transformation by tree-walking transducers with invisible pebbles. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 63–72. ACM, 2007. (Cited page 107)

[EHvB99] Joost Engelfriet, Hendrik Jan Hoogeboom, and Jan-Pascal van Best. Trips on trees. *Acta Cybernetica*, 14(1):51–64, 1999. (Cited page 105)

[Eil74] Samuel Eilenberg. *Automata, languages, and machines. Volume A*. Pure and applied mathematics. Academic Press, 1974. (Cited pages 9, 22, 78, and 84)

[EIM19] Joost Engelfriet, Kazuhiro Inaba, and Sebastian Maneth. Linear-bounded composition of tree-walking tree transducers: linear size increase and complexity. *Acta Informatica*, 2019. (Cited page 107)

[EM65] C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965. (Cited pages 9, 16, 78, 84, and 90)

[EM99] Joost Engelfriet and Sebastian Maneth. Macro tree transducers, attribute grammars, and MSO definable tree translations. *Information and Computation*, 154(1):34–91, 1999. (Cited page 107)

[EM02] Joost Engelfriet and Sebastian Maneth. Two-way finite state transducers with nested pebbles. In *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 234–244. Springer, 2002. (Cited page 104)

[Eng75] Joost Engelfriet. Bottom-up and top-down tree transformations - A comparison. *Mathematical Systems Theory*, 9(3):198–231, 1975. (Cited page 107)

[Eng81] Joost Engelfriet. Three hierarchies of transducers. *Mathematical systems theory*, 15(1):95–125, 1981. (Cited pages 18, 25, and 105)

[ERS80] Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. Tree transducers, L systems, and two-way machines. *Journal of Computer and System Sciences*, 20:150–202, 01 1980. (Cited page 18)

[ÉW10] Zoltán Ésik and Pascal Weil. Algebraic characterization of logically defined tree languages. *International Journal of Algebra and Computation (IJAC)*, 20(2):195–239, 2010. (Cited page 108)

[EY71] R.W. Ehrich and S.S. Yau. Two-way sequential transductions and stack automata. *Information and Control*, 18(5):404 – 446, 1971. (Cited pages 16 and 18)

[FGL16a] Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. Aperiodicity of rational functions is PSpace-complete. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, pages 13:1–13:15, 2016. (Cited page 12)

[FGL16b] Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. First-order definability of rational transductions: An algebraic approach. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16*, pages 387–396, 2016. (Cited page 12)

[FGL19] Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. Logical and algebraic characterizations of rational transductions. *Logical Methods in Computer Science*, 15(4), 2019. (Cited pages 12 and 89)

[FGLM18] Emmanuel Filiot, Olivier Gauwin, Nathan Lhote, and Anca Muscholl. On canonical models for rational functions over infinite words. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018*, volume 122 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. (Cited page 13)

[FGRS11] Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. Streamability of nested word transductions. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011*, volume 13 of *LIPIcs*, pages 312–324. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. (Cited page 12)

[FGRS13] Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. From two-way to one-way finite state transducers. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 468–477. IEEE Computer Society, 2013. (Cited pages 10, 12, 36, 39, and 44)

[FGRS19] Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. Streamability of nested word transductions. *Logical Methods in Computer Science*, 15(2), 2019. (Cited page 12)

[Fil15] Emmanuel Filiot. Logic-automata connections for transformations. In *Logic and Its Applications*, pages 30–57. Springer Berlin Heidelberg, 2015. (Cited pages 9, 16, 28, 29, 78, and 91)

[FJLW16] Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, pages 125:1–125:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. (Cited pages 19 and 103)

[FKT14] Emmanuel Filiot, Shankara Narayanan Krishna, and Ashutosh Trivedi. First-order definable string transformations. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014*, volume 29 of *LIPIcs*, pages 147–159. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. (Cited pages 80 and 108)

[FL15] Diego Figueira and Leonid Libkin. Synchronizing relations on words. *Theory of Computing Systems*, 57(2):287–318, 2015. (Cited page 19)

[FMR18] Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. A pattern logic for automata with outputs. In *Developments in Language Theory*, pages 304–317. Springer International Publishing, 2018. (Cited page 26)

[FMRT18] Emmanuel Filiot, Sebastian Maneth, Pierre-Alain Reynier, and Jean-Marc Talbot. Decision problems of tree transducers with origin. *Information and Computation*, 261:311–335, 2018. (Cited page 19)

[FR68] Patrick C. Fischer and Arnold L. Rosenberg. Multitape one-way nonwriting automata. *Journal of Computer and System Sciences*, 2(1):88 – 101, 1968. (Cited pages 10 and 23)

[FR16] Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *SIGLOG News*, 3(3):4–19, 2016. (Cited page 15)

[FR17] Emmanuel Filiot and Pierre-Alain Reynier. Copyful streaming string transducers. In *Reachability Problems - 11th International Workshop, RP 2017*, volume 10506 of *Lecture Notes in Computer Science*, pages 75–86. Springer, 2017. (Cited page 32)

[FRR+18] Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Visibly pushdown transducers. *Journal of Computer and System Sciences*, 97:147–181, 2018. (Cited pages 61, 69, and 73)

[FSM11] Sylvia Friese, Helmut Seidl, and Sebastian Maneth. Earliest normal form and minimization for bottom-up tree transducers. *International Journal of Foundations of Computer Science*, 22(7):1607–1623, 2011. (Cited page 108)

[FW65] N.J. Fine and H.S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16:109–114, 1965. (Cited page 43)

[GGP14] Viliam Geffert, Bruno Guillon, and Giovanni Pighizzini. Two-way automata making choices only at the endmarkers. *Information and Computation*, 239:71–86, 2014. (Cited page 105)

[GHI67] James N. Gray, Michael A. Harrison, and Oscar H. Ibarra. Two-way pushdown automata. *Information and Control*, 11(1):30 – 70, 1967. (Cited page 106)

[Gin66] Seymour Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, Inc., New York, NY, USA, 1966. (Cited pages 22 and 25)

[Gir86] Françoise Gire. Two decidability problems for infinite words. *Information Processing Letters*, 22(3):135–140, 1986. (Cited page 94)

[GKMP18] Bruno Guillon, Martin Kutrib, Andreas Malcher, and Luca Prigioniero. Reversible pushdown transducers. In *Developments in Language Theory - 22nd International Conference, DLT 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2018. (Cited page 108)

[GKS07] Martin Grohe, Christoph Koch, and Nicole Schweikardt. Tight lower bounds for query processing on streaming and external memory data. *Theoretical Computer Science*, 380:199–217, July 2007. (Cited page 61)

[GMPS17] Paul Gallot, Anca Muscholl, Gabriele Puppis, and Sylvain Salvati. On the decomposition of finite-valued streaming string transducers. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017*, volume 66 of *LIPIcs*, pages 34:1–34:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. (Cited pages 103 and 104)

[GMR20] Olivier Gauwin, Anca Muscholl, and Michael Raskin. Minimization of visibly pushdown automata is NP-complete. *Logical Methods in Computer Science*, Volume 16, Issue 1, 2020. (Cited page 61)

[GMS17] Nathan Grosshans, Pierre McKenzie, and Luc Segoufin. The power of programs over monoids in DA. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017*, volume 83 of *LIPIcs*, pages 2:1–2:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. (Cited page 108)

[GNT09] Olivier Gauwin, Joachim Niehren, and Sophie Tison. Earliest query answering for deterministic nested word automata. In *Fundamentals of Computation Theory, 17th International Symposium, FCT 2009*, volume 5699 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2009. (Cited page 11)

[GO14] Viliam Geffert and Alexander Okhotin. Transforming two-way alternating finite automata to one-way nondeterministic automata. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2014. (Cited page 41)

[GR63] Seymour Ginsburg and Gene F. Rose. Some recursively unsolvable problems in algol-like languages. *Journal of the ACM*, 10(1):29–47, January 1963. (Cited page 70)

[GR66] Seymour Ginsburg and Gene F. Rose. A characterization of machine mappings. *Canadian Journal of Mathematics*, 18:381–388, 1966. (Cited page 9)

[Gri68] T. V. Griffiths. The unsolvability of the equivalence problem for ε-free nondeterministic generalized machines. *Journal of the ACM*, 15(3):409–413, 1968. (Cited pages 10 and 23)

[Gui16a] Bruno Guillon. Input- or output-unary sweeping transducers are weaker than their 2-way counterparts. *RAIRO - Theoretical Informatics and Applications*, 50(4):275–294, 2016. (Cited page 24)

[Gui16b] Bruno Guillon. *Two-wayness: automata and transducers.* PhD thesis, Université Paris Diderot, Paris 7 and Università degli Studi di Milano, 2016. (Cited page 80)

[Gui18] Bruno Guillon. On nondeterministic two-way transducers. In *Tenth Workshop on Non-Classical Models of Automata and Applications, NCMA 2018*, pages 11–27. Österreichische Computer Gesellschaft, 2018. (Cited page 80)

[Gur82] Eitan M. Gurari. The equivalence problem for deterministic two-way sequential transducers is decidable. *SIAM Journal on Computing*, 11(3):448–452, 1982. (Cited page 25)

[Hib67] Thomas N. Hibbard. A generalization of context-free determinism. *Information and Control*, 11(1):196 – 238, 1967. (Cited page 104)

[HK91] Tero Harju and Juhani Karhumäki. The equivalence problem of multitape finite automata. *Theoretival Computer Science*, 78(2):347–355, 1991. (Cited page 18)

[HO07] Matthew Hague and C.-H. Luke Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. In *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007*, volume 4423 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2007. (Cited page 41)

[HU67] J. E. Hopcroft and J. D. Ullman. An approach to a unified theory of automata. *Bell System Technical Journal*, 46(8):1793–1829, 1967. (Cited pages 10 and 25)

[Hul15] Mans Hulden. From two-way to one-way finite automata - three regular expression-based methods. In *Implementation and Application of Automata - 20th International Conference, CIAA 2015*, volume 9223 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2015. (Cited page 41)

[Iba78] Oscar H. Ibarra. The unsolvability of the equivalence problem for epsilon-free NGSM's with unary input (output) alphabet and applications. *SIAM J. Comput.*, 7(4):524–532, 1978. (Cited pages 10, 23, and 24)

[JF18] Ismaël Jecker and Emmanuel Filiot. Multi-sequential word relations. *International Journal of Foundations of Computer Science*, 29(2):271–296, 2018. (Cited page 102)

[JO17] Galina Jirásková and Alexander Okhotin. On the state complexity of operations on two-way finite automata. *Information and Computation*, 253:36–63, 2017. (Cited page 37)

[Kap05] Christos A. Kapoutsis. Removing bidirectionality from nondeterministic finite automata. In *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Proceedings*, volume 3618 of *Lecture Notes in Computer Science*, pages 544–555. Springer, 2005. (Cited pages 36 and 38)

[KKM+14] Makoto Kanazawa, Gregory M. Kobele, Jens Michaelis, Sylvain Salvati, and Ryo Yoshinaka. The failure of the strong pumping lemma for multiple context-free languages. *Theory of Computing Systems*, 55(1):250–278, 2014. (Cited page 102)

[KMP14] Martin Kutrib, Andreas Malcher, and Giovanni Pighizzini. Oblivious two-way finite automata: Decidability and complexity. *Information and Compututation*, 237:294–302, 2014. (Cited page 36)

[KO11] Michal Kunc and Alexander Okhotin. Describing periodicity in two-way deterministic finite automata using transformation semigroups. In *Developments in Language Theory - 15th International Conference, DLT 2011. Proceedings*, volume 6795 of *Lecture Notes in Computer Science*, pages 324–336. Springer, 2011. (Cited page 37)

[KO12] Michal Kunc and Alexander Okhotin. State complexity of operations on two-way finite automata over a unary alphabet. *Theoretical Computer Science*, 449:106–118, 2012. (Cited page 37)

[Kob69] Kojiro Kobayashi. Classification of formal languages by functional binary transductions. *Information and Control*, 15(1):95–109, 1969. (Cited page 103)

[Kop16] Eryk Kopczyński. Invisible pushdown languages. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'16*, pages 867–872. ACM, 2016. (Cited page 109)

[KP12] Ondrej Klíma and Libor Polák. On biautomata. *RAIRO - Theoretical Informatics and Applications*, 46(4):573–592, 2012. (Cited page 80)

[KP15] Christos A. Kapoutsis and Giovanni Pighizzini. Two-way automata characterizations of l/poly versus NL. *Theory of Computing Systems*, 56(4):662–685, 2015. (Cited page 105)

[Lat77] Michel Latteux. EDT0L, systèmes ultralinéaires et opérateurs associés, 1977. T.R. 100, Université Lille. (Cited page 18)

[Led13] Jérémy Ledent. Internship report - streaming string transducers, 2013. Université de Bordeaux. (Cited pages 31, 32, and 60)

[Lho20] Nathan Lhote. Pebble minimization of polyregular functions. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'20 (to appear)*. ACM, 2020. (Cited page 104)

[LMSV99] Clemens Lautemann, Pierre McKenzie, Thomas Schwentick, and Heribert Vollmer. The descriptive complexity approach to LOGCFL. In *16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 of *LNCS*, pages 444–454. Springer, 1999. (Cited page 79)

[LMT19] Théodore Lopez, Benjamin Monmege, and Jean-Marc Talbot. Determinisation of finitely-ambiguous copyless cost register automata. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, volume 138 of *LIPIcs*, pages 75:1–75:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited page 109)

[Lom02] Sylvain Lombardy. On the construction of reversible automata for reversible languages. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 170–182. Springer, 2002. (Cited page 108)

[Lom16] Sylvain Lombardy. Two-way representations and weighted automata. *RAIRO - Theoretical Informatics and Applications*, 50(4):331–350, 2016. (Cited page 109)

[LPP17a] Giovanna J. Lavado, Giovanni Pighizzini, and Luca Prigioniero. Minimal and reduced reversible automata. *Journal of Automata, Languages and Combinatorics*, 22(1-3):145–168, 2017. (Cited page 108)

[LPP17b] Giovanna J. Lavado, Giovanni Pighizzini, and Luca Prigioniero. Weakly and strongly irreversible regular languages. In *Proceedings 15th International Conference on Automata and Formal Languages, AFL 2017*, volume 252 of *EPTCS*, pages 143–156, 2017. (Cited page 108)

[LS06] Sylvain Lombardy and Jacques Sakarovitch. Sequential? *Theoretical Computer Science*, 356(1-2):224–244, 2006. (Cited page 23)

[LS19] Christof Löding and Christopher Spinrath. Decision Problems for Subclasses of Rational Relations over Finite and Infinite Words. *Discrete Mathematics & Theoretical Computer Science*, Vol. 21 no. 3 , 2019. (Cited pages 61 and 107)

[LW17] Christof Löding and Sarah Winter. Synthesis of deterministic top-down tree transducers from automatic tree relations. *Information and Computation*, 253:336–354, 2017. (Cited page 107)

[Meh80] Kurt Mehlhorn. Pebbling moutain ranges and its application of dcfl-recognition. In *Automata, Languages and Programming, 7th Colloquium, ICALP 1980. Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980. (Cited page 61)

[MF71] A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *12th Annual Symposium on Switching and Automata Theory (SWAT 1971)*, pages 188 – 191, 1971. (Cited page 36)

[Moo71] Frank R. Moore. On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Trans. Computers*, 20(10):1211–1214, 1971. (Cited page 36)

[MP71] Robert McNaughton and Seymour Papert. *Counter-Free Automata*. Number 65 in M.I.T. Press research monographs. The MIT Press, 1971. (Cited pages 77 and 92)

[MP19a] Anca Muscholl and Gabriele Puppis. Equivalence of finite-valued streaming string transducers is decidable. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 122:1–122:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited pages 104 and 108)

[MP19b] Anca Muscholl and Gabriele Puppis. The many facets of string transducers (invited talk). In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPIcs*, pages 2:1–2:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited pages 15, 25, and 26)

[MS92] A. Muchnik and A. L. Semenov. Automata on infinite objects, monadic theories, and complexity, 1992. (Cited page 41)

[MS18] Sebastian Maneth and Helmut Seidl. Balancedness of MSO transductions in polynomial time. *Information Processing Letters*, 133:26–32, 2018. (Cited page 107)

[MSTV06] Pierre McKenzie, Thomas Schwentick, Denis Thérien, and Heribert Vollmer. The many faces of a translation. *Journal of Computer and System Science*, 72(1):163–179, 2006. (Cited pages 79 and 108)

[MV09] P. Madhusudan and Mahesh Viswanathan. Query automata for nested words. In *Mathematical Foundations of Computer Science 2009, 34th International Symposium, MFCS 2009. Proceedings*, volume 5734 of *Lecture Notes in Computer Science*, pages 561–573. Springer, 2009. (Cited page 106)

[Myh57] John Myhill. Finite automata and the representation of events. In *Fundamental Concepts in the Theory of Systems (WADC publication; ASTIA No. AD155741)*, page 112–137. Wright Patterson AFB, Ohio, 1957. (Cited pages 9, 11, and 77)

[Ner63] Anil Nerode. Linear automaton transformations. *Journal of Symbolic Logic*, 28(2):173–174, 1963. (Cited pages 9, 11, and 77)

[Niv68] Maurice Nivat. Transduction des langages de Chomsky. *Annales de l'Institut Fourier*, 28:339–455, 1968. (Cited page 9)

[Per84] Dominique Perrin. Recent results on automata and infinite words. In *Mathematical Foundations of Computer Science 1984, MFCS'84*, volume 176 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 1984. (Cited page 79)

[Pet98] Holger Petersen. The head hierarchy for oblivious finite automata with polynomial advice collapses. In *Mathematical Foundations of Computer Science 1998*, pages 296–304, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. (Cited page 36)

[Pig19] Giovanni Pighizzini. Limited automata: Properties, complexity and variants. In *Descriptional Complexity of Formal Systems - 21st IFIP WG 1.02 International Conference, DCFS 2019, Proceedings*, volume 11612 of *Lecture Notes in Computer Science*, pages 57–73. Springer, 2019. (Cited page 104)

[PP04] Dominique Perrin and Jean-Éric Pin. *Infinite words - automata, semigroups, logic and games*, volume 141 of *Pure and applied mathematics series*. Elsevier Morgan Kaufmann, 2004. (Cited page 79)

[PP13] Giovanni Pighizzini and Andrea Pisoni. Limited automata and regular languages. In *Descriptional Complexity of Formal Systems - 15th International Workshop, DCFS 2013. Proceedings*, volume 8031 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 2013. (Cited page 104)

[Pri02] Christophe Prieur. How to decide continuity of rational functions on infinite words. *Theoretical Computer Science*, 276(1-2):445–447, 2002. (Cited page 94)

[PS99] Maryse Pelletier and Jacques Sakarovitch. On the representation of finite deterministic 2-tape automata. *Theoretical Computer Science*, 225(1-2):1–63, 1999. (Cited page 18)

[PS05] Jean-Éric Pin and Howard Straubing. Some results on $\mathcal{C}$-varieties. *ITA*, 39(1):239–262, 2005. (Cited page 81)

[PZ14] Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 75:1–75:10. ACM, 2014. (Cited page 108)

[PZ18] Thomas Place and Marc Zeitoun. The covering problem. *Logical Methods in Computer Science*, 14(3), 2018. (Cited page 109)

[PZ19] Thomas Place and Marc Zeitoun. Going higher in first-order quantifier alternation hierarchies on words. *Journal of the ACM*, 66(2):12:1–12:65, 2019. (Cited page 109)

[Raj72] Vaclav Rajlich. Absolutely parallel grammars and two-way finite-state transducers. *Journal of Computer and System Sciences*, 6(4):324 – 342, 1972. (Cited page 18)

[Rey15] Pierre-Alain Reynier. Contributions to timed systems and transducers. Habilitation à diriger des recherches (habilitation thesis). Laboratoire d'informatique fondamentale de Marseille, Aix-Marseille Université, France, 2015. (Cited pages 103 and 105)

[Roz85] Brigitte Rozoy. About two-way transducers. In *Fundamentals of Computation Theory, FCT '85*, volume 199 of *Lecture Notes in Computer Science*, pages 371–379. Springer, 1985. (Cited page 18)

[Roz86] Brigitte Rozoy. Outils et résultats pour les transducteurs boustrophédons. *ITA*, 20(3):221–249, 1986. (Cited pages 36, 102, and 107)

[Roz87] Brigitte Rozoy. The Dyck language $D_1'^*$ is not generated by any matric grammar of finite index. *Information and Computation*, 74(1):64–89, 1987. (Cited page 18)

[RS59] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959. (Cited pages 9, 12, 15, 35, 37, 38, 39, and 101)

[RS91] Christophe Reutenauer and Marcel-Paul Schützenberger. Minimization of rational word functions. *SIAM Journal of Computing*, 20(4):669–685, 1991. (Cited pages 9, 10, 78, 85, 87, 88, and 98)

[RT16] Pierre-Alain Reynier and Jean-Marc Talbot. Visibly pushdown transducers with well-nested outputs. *International Journal of Foundations of Computer Science*, 27(2):235–258, 2016. (Cited page 107)

[RV19] Pierre-Alain Reynier and Didier Villevalois. Sequentiality of string-to-context transducers. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 128:1–128:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited page 18)

[Saa19] Aleksi Saarela. Word equations with kth powers of variables. *Journal of Combinatorial Theory, Series A*, 165:15 – 31, 2019. (Cited page 75)

[Sak09] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009. (Cited pages 10, 22, and 41)

[Sch61] Marcel-Paul Schützenberger. A remark on finite transducers. *Information and Control*, 4(2-3):185–196, 1961. (Cited pages 9, 10, 17, 78, and 84)

[Sch65] Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965. (Cited pages 77 and 92)

[Sch77] Marcel-Paul Schützenberger. Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4(1):47 – 57, 1977. (Cited pages 23, 25, 78, and 80)

[SdS10] Jacques Sakarovitch and Rodrigo de Souza. Lexicographic decomposition of k-valued transducers. *Theory of Computing Systems*, 47(3):758–785, 2010. (Cited page 104)

[Sem84] A. L. Semenov. Decidability of monadic theories. In *Mathematical Foundations of Computer Science 1984*, pages 162–175. Springer Berlin Heidelberg, 1984. (Cited page 41)

[She59] J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959. (Cited pages 12, 15, 16, 35, and 37)

[Sim90] Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990. (Cited pages 54, 56, and 66)

[Sip80] Michael Sipser. Lower bounds on the size of sweeping automata. *Journal of Computer and System Sciences*, 21(2):195 – 202, 1980. (Cited pages 15 and 36)

[Smi14] Tim Smith. A pumping lemma for two-way finite transducers. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014*, volume 8634 of *Lecture Notes in Computer Science*, pages 523–534. Springer, 2014. (Cited pages 18, 36, and 102)

[SMK18] Helmut Seidl, Sebastian Maneth, and Gregor Kemper. Equivalence of deterministic top-down tree-to-string transducers is decidable. *Journal of the ACM*, 65(4):21:1–21:30, 2018. (Cited pages 10 and 108)

[Srb09] Jirí Srba. Beyond language equivalence on visibly pushdown automata. *Logical Methods in Computer Science*, 5(1:2):1–22, 2009. (Cited pages 61 and 107)

[SS78] William J. Sakoda and Michael Sipser. Nondeterminism and the size of two way finite automata. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC)*, page 275–286. Association for Computing Machinery, 1978. (Cited pages 10, 15, 36, and 105)

[SS07] Luc Segoufin and Cristina Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *Proceedings of the 11th International Conference on Database Theory (ICDT'07)*, pages 299–313. Springer Berlin Heidelberg, 2007. (Cited page 60)

[Ste67] Richard E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11(3):323–340, 1967. (Cited pages 61 and 70)

[Ste85] Jacques Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985. (Cited page 90)

[Str94] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Progress in Computer Science and Applied Series. Birkhäuser, 1994. (Cited pages 27 and 77)

[SY06] Nicolae Santean and Sheng Yu. Nondeterministic bimachines and rational relations with finite codomain. *Fundamenta Informaticae*, 73(1-2):237–264, 2006. (Cited page 79)

[Tra61] Boris A Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140(326-329):122–123, 1961. (Cited pages 9, 15, 27, and 77)

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 234–240, 1998. (Cited page 82)

[Val75] Leslie G. Valiant. Regularity and related problems for deterministic pushdown automata. *Journal of the ACM*, 22(1):1–10, 1975. (Cited pages 61 and 70)

[Var89] Moshe Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, 1989. (Cited page 40)

[Web96] Andreas Weber. Decomposing A k-valued transducer into k unambiguous ones. *Informatique Théorique et Applications*, 30(5):379–413, 1996. (Cited page 104)

[Wil93] Thomas Wilke. An algebraic theory for regular languages of finite and infinite words. *IJAC*, 3(4):447–490, 1993. (Cited page 79)

[Wil16] Thomas Wilke. Past, present, and infinite future. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, pages 95:1–95:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. (Cited page 79)

[WK95] A. Weber and R. Klemm. Economy of description for single-valued transducers. *Information and Computation*, 118(2):327 – 340, 1995. (Cited pages 26, 70, and 73)

[WW86] K. Wagner and G. Wechsung. *Computational Complexity*, volume 21 of *Mathematics and its Applications*. Springer, 1986. (Cited page 104)

[YY19] Di-De Yen and Hsu-Chun Yen. Characterizing the valuedness of two-way finite transducers. In *Developments in Language Theory - 23rd International Conference, DLT 2019, Proceedings*, volume 11647 of *Lecture Notes in Computer Science*, pages 100–112. Springer, 2019. (Cited page 104)

[ZC05] Afra Zomorodian and Gunnar E. Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005. (Cited page 39)